Asistente accesible en Android para la identificación de billetes con cambio de divisa en tiempo real
Accessible assistant on Android to identify banknotes with currency exchange in real time

Trabajo de Fin de Máster
Curso 2019–2020

Autor
Francis Andrés Cachago Tandazo

Directores
María Guijarro Mata-García
Joaquín Recas Piorno

Máster en Internet de las Cosas
Facultad de Informática
Universidad Complutense de Madrid

Asistente accesible en Android para la identificación de billetes con cambio de divisa en tiempo real
Accessible assistant on Android to identify banknotes with currency exchange in real time

Trabajo de Fin de Máster en Internet de las Cosas
Departamento de Arquitectura de Computadores y Automática

Autor
Francis Andrés Cachago Tandazo

Directores
María Guijarro Mata-García
Joaquín Recas Piorno

Convocatoria: *Septiembre 2020*
Calificación: *7.5*

Máster en Internet de las Cosas
Facultad de Informática
Universidad Complutense de Madrid

23 de Septiembre de 2020

Dedicatoria

Dedico este trabajo a mis padres porque estuvieron a mi lado siempre, en los buenos y malos momentos sin importar lo que la vida nos ponga en frente. De igual manera se lo dedico a Dios quien me permite vivir el día a día cumpliendo mis sueños.

Agradecimientos

Quiero agradecer a Dios quien me puso en este camino y ha sido una constante guía, permitiéndome conocer muy buenas personas, tener grandiosos amigos como Lidia, Luis, Adrián, Mathias y muchos más, y estar acompañado de excelentes padres para lograr mis metas. De igual manera agradezco a mi familia que de una u otra manera me acompañan y me brindan su apoyo desde lejos.

También agradezco a mis tutores, María y Joaquín, quienes me han orientado por medio de sus comentarios y consejos para desarrollar y culminar este proyecto. De igual forma gracias a la Universidad Complutense y a sus profesionales quienes me ayudaron a adquirir nuevos conocimientos a lo largo de este máster.

Resumen

Asistente accesible en Android para la identificación de billetes con cambio de divisa en tiempo real

Actualmente vivimos rodeados de tecnología y dispositivos electrónicos que evolucionan constantemente, como por ejemplo un teléfono móvil el cual al integrar sus distintos periféricos, su capacidad de procesamiento y conectividad a internet permite realizar varias tareas que facilitan el día a día de las personas y más de aquellas que tienen discapacidades, como por ejemplo las personas ciegas.

El presente trabajo implementa varios conceptos del Internet de las Cosas (IoT) con el objetivo de facilitar la tarea de identificación de billetes euros y dólares, a personas ciegas por medio de un asistente accesible para teléfonos móviles Android. El asistente también informa del cambio de divisa en tiempo real y toda esta información es desplegada en texto; con ayuda herramientas de accesibilidad como TalkBack se transforma ese texto en voz, facilitando así la interacción del usuario con la aplicación.

Para la identificación de billetes se crea un servicio en AWS Lambda que trabaja bajo peticiones. Este servicio utiliza un modelo de redes neuronales desarrollado con herramientas de visión con la librería fast.ai y PyTorch, para el reconocimiento de imágenes. El modelo es entrenado, posteriormente puesto a prueba en dos escenarios y finalmente implementado. Por otro lado, para conocer la información sobre el cambio de divisas, se ha tomado un servicio en la nube de una fuente de datos abiertos por medio de una API Rest.

Estos servicios son integrados en una aplicación la cual tiene como característica ser desarrollada en Apache Cordova con un enfoque web, permitiendo así agregar varios microservicios que funcionen a la par. Con Cordova también se puede acceder a los distintos sensores, en este caso la cámara que se utilizar para hacer fotos a los distintos billetes y luego enviar esta información para que sea procesada por los servicios en la nube.

Palabras clave

Internet de las Cosas, Inteligencia Artificial, Aplicaciones Multiplataforma, Microservicios, Apache Cordova, AWS Lambda, Redes Neuronales, Computación sin servidor, Python, HMTL5, CSS3, Javascript

Abstract

Accessible assistant on Android to identify banknotes with currency exchange in real time

Currently we live surrounded by technology and electronic devices in constant evolution, such as a mobile phone that by integrating its different peripherals, its processing capacity and internet connectivity allows us to perform various tasks that facilitate people's day-to-day life and more than they have disabilities, like blind people.

The present work implements several Internet of Things (IoT) concepts with the aim of facilitating the task of identifying euro and dollar bills for blind people through an accessible assistant for Android mobile phones. The assistant also reports the currency exchange in real time and all this information is displayed in text; With the help of accessibility tools like TalkBack, this text is transformed into speech, which facilitates user interaction with the application.

For the identification of banknotes, a service is created in AWS Lambda that works under requests. This service uses a neural network model developed with vision tools with the fast.ai and PyTorch library, for image recognition. The model is trained, subsequently tested in two scenarios, and finally implemented. On the other hand, to know the information about currency exchange, a cloud service has been taken from an open data source through a Rest API.

These services are integrated into an application that has the characteristic of being developed in Apache Cordova with a web approach, which allows adding several microservices that work together. With Cordova you can also access the different sensors, in this case the camera that will be used to take photos of the different banknotes and then send this information to be processed by cloud services.

Keywords

Internet of Things, Artificial intelligence, cross-platform applications, microservices, Apache Cordova, AWS Lambda, Neural Networks, Serverless Computing, Python, HMTL5, CSS3, Javascript

Índice

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

"Let them rise to the challenge of Sustainable Development Goals and act, not out of self interest, but out of common interest. I am very aware of the preciousness of time. Seize the moment, act now."
— Stephen Hawking

En la actualidad vivimos en un mundo en el cual el desarrollo y la presencia de la tecnología en nuestras vidas aumenta minuto a minuto. El objetivo de querer comunicarse entre distintas partes del planeta a través de dispositivos tecnológicos en tiempo real ha dado como resultado un mundo totalmente conectado en el que prácticamente en cualquier lugar se tiene acceso a la mayor red de datos, internet.

Específicamente, el Internet de las Cosas (IoT) evoluciona de tal manera que, para 2024, se calcula que habrá 4000 millones de dispositivos conectados a Internet únicamente en entornos IoT. Esta cifra ya de por sí, indica una enorme cantidad de dispositivos conectados, si añadimos otros tantos como teléfonos móviles, la cifra prevista asciende hasta los 8900 millones (Cerwall, 2015).

Por IoT se entiende aquellas soluciones basadas en procesar información adquirida localmente mediante sensores, como por ejemplo cámaras, en un sistema inteligente. Estos sistemas frecuentemente centralizan la información, gestionando datos de varios dispositivos lo que permite decidir combinarlos entre ellos o con datos procedentes de otras fuentes.

Esto ha permitido el desarrollo de un sinnúmero de aplicaciones, algunas de ellas enfocadas a la ayuda social; por mencionar algunas están TapTapSee que es utilizada para reconocimiento de objetos, Boop Light Detector permite detectar si la luz de una habitación sigue encendida, NantMobile Money Reader sirve para identificar el valor de cualquier billete de curso legal de distintos países, entre otras. De esta manera, personas con discapacidades como por ejemplo la visual, han ido integrándose más a la sociedad gracias al avance de la tecnología y al uso de aplicaciones como las mencionadas.

1.1. Motivación

De acuerdo a la Organización Mundial de la Salud (OMS), a nivel mundial se calcula que aproximadamente 1300 millones de personas viven con alguna forma de deficiencia de la visión de lejos o de cerca. Con respecto a la visión de lejos, 188,5 millones de personas tienen una deficiencia visual moderada, 217 millones tienen una deficiencia visual de moderada a grave y 36 millones son ciegas. Por otro lado, 826 millones de personas padecen una

deficiencia de la visión de cerca (OMS, 2018). En España, de acuerdo a la información de la Organización Nacional de Ciegos de España (ONCE), 72 231 personas están afiliadas y registradas con deficiencia visual, de las cuales el 14.24 % tiene ceguera y el resto cuenta con una discapacidad visual más leve como se observa en la Figura 1.1 (ONCE, 2019).

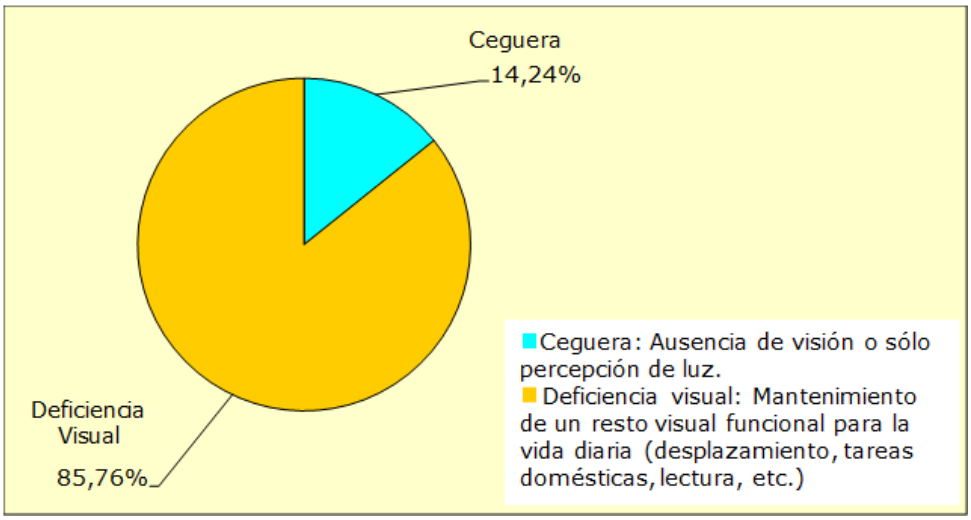

Figura 1.1: Distribución del total de afiliados a la ONCE por tipo de pérdida. Fuente: ONCE (2019).

Las personas con discapacidad visual pueden encontrar problemas a diario como por ejemplo a la hora de distinguir los billetes, sobretodo para el caso de divisas de otros países al momento de realizar compras o el simple hecho de tener dinero a la mano.

De acuerdo a un artículo publicado por la ONCE (2016), existen algunas aplicaciones con buena aceptación por parte de personas con discapacidad visual ya que de alguna u otra forma ayudan en su día a día; NantMobile Money Reader destaca entre ellas ya que es una aplicación para el reconocimiento de dinero para dispositivos móviles iOS. Esta aplicación informa del valor de cualquier billete de curso legal de distintos países y está disponible en App Store por un costo de 13.99 €.

Se debe mencionar que algunas aplicaciones móviles, no están diseñadas para personas invidentes como usuarios finales debido a que no cuentan con características de accesibilidad para el uso de las mismas. Es decir, el manejo o navegación es tedioso y aplicaciones que brindan estas características suelen tener un costo promedio de 10 €.

Tomando en cuenta lo antes mencionado, se ha planteado la idea de desarrollar un asistente que permitirá identificar billetes, específicamente euros y dólares americanos, junto con su valor para dispositivos móviles con sistema operativo Android el cual es enfocado para personas invidentes. Por lo que la aplicación cubrirá requerimientos mínimos de accesibilidad para los usuarios finales. Además, una vez identificado el valor del billete, la aplicación permitirá saber el valor de cambio de divisa actual y como diferencia notable de otras aplicaciones será gratuita, además de poder ser adaptable a otras plataformas o sistemas operativos.

1.2. Objetivos

El principal objetivo de este proyecto es desarrollar un asistente accesible para móviles con sistema operativo Android, que permita a personas ciegas identificar billetes (dólares

o euros) con su cambio de divisa en tiempo real.

Además, se presentan otros objetivos relevantes para la implementación de este proyecto, los cuales se destacan a continuación:

- Estudiar y documentar la situación actual de aplicaciones enfocadas al reconocimiento de billetes.
- Seleccionar y describir el dispositivo a ser utilizado, que cumpla las características para trabajar en un ambiente IoT.
- Estudiar las tecnologías que pueden emplearse para desarrollar aplicaciones IoT para dispositivos móviles y que permitan escalabilidad.
- Investigar, desarrollar e implementar un modelo con inteligencia artificial para el reconocimiento de billetes, euros y dólares americanos, por medio de imágenes.
- Investigar y añadir un servicio para el cambio de divisas de acuerdo a información del mercado en tiempo real.
- Decidir qué dispositivos o servicios van a realizar el cómputo de la información.
- Seleccionar qué infraestructura o servicio va a implementar el modelo de clasificación.
- Integrar las tecnologías utilizadas para que funcionen en conjunto.
- Determinar los usuarios finales para la aplicación.
- Desarrollar e implementar la aplicación para que funcione en el dispositivo IoT con sistema operativo Android.

1.3. Plan de trabajo

Para realizar el proyecto se ha seguido el siguiente plan de trabajo:

- **Análisis del problema.** En esta primera fase se trata de entender el problema, mediante la recopilación de distintas fuentes de información y estado de arte actual, vinculados a la realización del proyecto así como los requerimientos de los potenciales usuarios del mismo para fijar la funcionalidad del proyecto a implementar.
- **Diseño de la arquitectura del software.** Para esta fase se decidirán los módulos en los que ha de dividirse la aplicación, así como las tecnologías necesarias para el desarrollo de cada uno de ellos.
- **Estudio de las tecnologías a utilizar.** Con la arquitectura diseñada, se tendrá que estudiar y configurar las tecnologías necesarias para hacerlo.
- **Desarrollo del software.** En esta etapa se implementarán y unirán todos los módulos resultantes del estudio de la tercera fase.
- **Implementación.** En esta fase se desplegará la aplicación para que funcione en el dispositivo IoT seleccionado y permita interactuar con los usuarios finales, determinando qué mejoras pueden ser implementadas para trabajos futuros.
- **Redacción de la memoria y documentación del proyecto.** Esta fase se realiza de manera paralela a todas las anteriores, de esta manera se colectará toda la información adquirida durante el desarrollo para la redacción de la documentación.

1.4. Estructura del documento

Esta sección presenta una visión de alto nivel de este documento que está organizado en distintos capítulos, con la siguiente estructura.

El Capítulo 1 es la introducción del proyecto y describe la motivación, los objetivos y la estructura del mismo.

En el Capítulo 2 se ilustra el estado del arte sobre el proyecto que en una primera parte se describen conceptos en los cuales se sustenta este proyecto y se requieren conocer. También se presenta a los usuarios finales y cuan importante es una aplicación con el enfoque accesible, de cero costo, desplegable en otras plataformas y con inteligencia propia. Finalmente, se describen proyectos similares al reconocimiento de billetes con sus características.

El Capítulo 3 presenta los distintos recursos, tecnologías y plataformas utilizados en base a los conceptos vistos en el Capitulo 2. Todo esto permite formar la infraestructura IoT del proyecto, de esta manera en la parte final se describe como están asociados los servicios en la nube con el cliente que será la aplicación utilizada por los usuarios objetivos.

En el Capítulo 4 se describe la implementación del proyecto, mostrando como fue desarrollado todo el proyecto desde los requerimientos iniciales como tecnologías, plataformas, etc., hasta los recursos implementados para desplegar el microservicio de identificación de billetes. el modelo de clasificación y cómo se obtiene la información del cambio de divisas. Además, se presenta como fue diseñada la aplicación para el dispositivo móvil con sistema operativo Android que será utilizado por personas ciegas quienes son el usuario objetivo de este poryecto.

El Capítulo 5 presenta un resumen del trabajo realizado y lo logrado, comentando las conclusiones obtenidas del desarrollo, los problemas encontrados y propuestas para posibles trabajos futuros.

Los Capítulos 6 y 7 son presentados en inglés. Estos contienen la misma información del Capítulo 1 y 5, respectivamente.

Capítulo 2

Estado de la Cuestión

"En principio, la investigación necesita más cabezas que medios."
— Severo Ochoa

En este capítulo se presenta los principales conceptos que se requiere conocer para el desarrollo de este proyecto y que permiten la implementación del mismo. También se presenta las características que deben tener las aplicaciones móviles para ser accesibles para personas ciegas quienes son los usuarios finales que igualmente son descritos. Por último, se destacan varias aplicaciones existentes relacionados al reconocimiento de billetes, al igual que los métodos y herramientas respecto al desarrollo de modelos con inteligencia artificial.

2.1. Internet de las Cosas

En la actualidad, es más común que objetos de la vida cotidiana tengan la capacidad de conectarse a Internet. Estos objetos inteligentes conectados a Internet constituyen el llamado "Internet de las cosas" (IoT). Tal es así, que sin darnos cuenta, a nuestro alrededor ahora se encuentra una malla invisible de comunicación de dispositivos integrados en aparentemente todo, revolucionando la interacción de los seres humanos con el mundo.

En IoT, una variedad de objetos se comunican y colaboran mutuamente, dando como resultado nuevas aplicaciones y servicios con valor añadido, que dan paso a un entorno inteligente ya que los sistemas podrán trabajar de forma autónoma y adecuada, reconociendo eventos y cambios.

El objetivo de esta nueva tendencia es romper las barreras físicas entre dispositivos, de manera que la comunicación entre los mismos esté menos limitada y que no solo se puedan conectar máquinas sino dispositivos de distintos tamaños y utilidades, como vehículos, sistemas industriales, personas, equipos médicos, entre otros. Lo que da como resultado una mejora notable en la calidad de vida de las personas y empresas (Caschetto, 2019).

Por último, IoT se ha visto impulsado por el surgimiento de la computación en la nube, con lo que se aprovechan los recursos informáticos conectados en red, logrando así el procesamiento, gestión y almacenamiento de datos. Así, dispositivos de pequeño tamaño pueden interactuar con sistemas más potentes y mejorar las capacidades analíticas y de control (Caschetto, 2019).

2.1.1. Dispositivos IoT

Los dispositivos IoT han llegado a la vida de todos de una manera abrumadora. Hoy en día, es muy raro y poco común encontrar algún objeto que no permita conectarse a

Internet. Además, vienen para hacer más fácil la vida al usuario, permitiendo un control, gestión de forma remota y/o monitoreo; todo esto en tiempo real.

Por lo tanto, un dispositivo IoT consiste en un objeto al que se le ha dotado de conexión a Internet y cierta inteligencia software, sobre el que se pueden medir parámetros físicos o actuar remotamente y por ende permite generar un ecosistema de servicios alrededor del mismo. Este ecosistema está destinado a generar valor transformando la experiencia del usuario (para los negocios Cámara de Valencia, 2018).

Ante esto, puede surgir la siguiente pregunta: ¿un teléfono móvil puede ser considerado como dispositivo IoT? En este punto, hay que decir que no hay que tirar mucho de imaginación. La respuesta es obvia, por supuesto que sí, ya que incluye un procesador informático y un sistema de hardware asociado que permiten que el teléfono presente una interfaz gráfica, ejecute un sistema operativo, se conecte al Internet y ejecute aplicaciones, entre otras tareas. Además, cuenta con sensores acoplados como son el acelerómetro, el giroscopio, la cámara, el GPS, etc.

2.2. Cloud Computing

Cloud computing o simplemente "la nube", es la oferta de recursos de computación bajo demanda que van desde aplicaciones, plataformas o infraestructuras por todo internet como se observa en la Figura 2.1, que por lo general aplica un sistema de pago por uso y algunos otras con consumos gratuitos hasta ciertos límites.

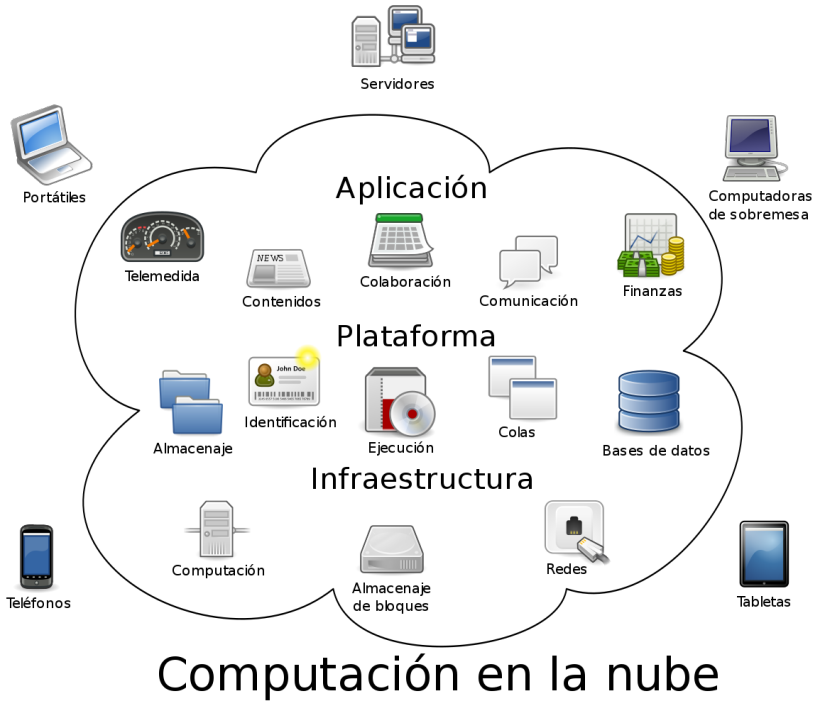

Figura 2.1: Diagrama de Cloud Computing. Fuente: Wikipedia (2020)

El uso de la palabra "nube" para definir a estas tecnologías tiene ciertas implicaciones. Esencialmente, la nube hace referencia a dos conceptos (Sosinsky, 2011):

- *Abstracción:* Las tareas se ejecutan en sistemas físicos que no están identificados (más

allá de referencias). Además, el acceso a estas tareas es posible desde cualquier lugar y dispositivo.

- *Virtualización:* los recursos se reúnen y se comparten desde un fondo común (los centros de datos). Existe la posibilidad de multi arrendamiento y la escalabilidad es sencilla y ágil.

El concepto de cloud computing en este proyecto es utilizado porque a través de la plataforma AWS que es descrita en la sección 3.2, se pueden realizar distintas tareas como por ejemplo desplegar microservicios, almacenar contenido, desarrollar códigos sin preocuparse por instalar las distintas dependencias, entre otras.

2.2.1. Características esenciales

- **Auto servicio bajo demanda:** Los usuarios pueden aprovisionar recursos y usarlos sin la interacción humana del proveedor de servicios.
- **Amplio acceso a la red:** Recursos están disponibles a través de la red y pueden ser accedidos por diversas plataformas de clientes.
- **Multi-arrendamiento y agrupación de recursos:**

  Varios clientes pueden compartir la misma infraestructura y aplicaciones con seguridad y privacidad.

  Múltiples clientes son atendidos desde los mismos recursos físicos.

- **Rápida elasticidad y escalabilidad:**

  Adquiere y elimina recursos de forma automática y rápida cuando sea necesario.

  Escala rápida y fácilmente en función de la demanda.

- **Servicio medido:** Los usuarios pagan por lo que han usado.

2.3. Serverless Computing

Las arquitecturas Serverless son diseños de aplicaciones que incorporan servicios "Backend as a Service" (BaaS) de terceros y/o que incluyen código personalizado ejecutado en contenedores administrados y efímeros en una plataforma de "Funciones como servicio" (FaaS) (Roberts, 2018). El uso de este concepto tiene como ventaja despreocupar al desarrollador de la gestión de la infraestructura sobre la cual se arropará y ejecutará el servicio (función), permitiendo centrarse en la funcionalidad, por lo que el ciclo de desarrollo se verá simplificado.

El servicio AWS Lambda empleado para este proyecto, trabaja con el concepto Serverless computing, específicamente FaaS porque permitirá desarrollar y ejecutar un código en la nube, incluyendo otros recursos de AWS, todo esto con el fin de separar el procesamiento del lado del dispositivo IoT. Pero sin la necesidad de montar un servidor con todas las dependencias necesarias para la ejecución de dicho código ya que todo esto es administrado por AWS.

Serverless también puede significar aplicaciones donde *la lógica del lado del servidor todavía es escrita por el desarrollador de la aplicación,* pero, a diferencia de las arquitecturas tradicionales, se ejecuta en contenedores de cómputo sin estado que son disparados por

eventos, efímeros (pueden durar solo una invocación) y completamente administrados por un tercero. Una forma de pensar en esto es "Funciones como servicio" o "FaaS" (Roberts, 2018). **AWS Lambda** es una de las implementaciones más populares de una plataforma de funciones como servicio en la actualidad.

2.3.1. Desembalaje de "Función como servicio" (FaaS)

Para poder entender mejor cómo es el funcionamiento de FaaS, se ha tomado el ejemplo presentado por Mike Roberts (2018) sobre AWS Lambda basado en la presentación de AWS (AWS, 2020b) y descrita en la sección 3.2.3 ya que con este recurso se implementa el código que permite identificar el billete capturado por la aplicación.

- *AWS Lambda permite ejecutar código sin aprovisionar ni administrar servidores.*

  Fundamentalmente, FaaS se trata de ejecutar código de back-end sin administrar sus propios sistemas de servidor o sus propias aplicaciones de servidor de larga duración. Las aplicaciones de servidor de larga duración, es una diferencia clave cuando se compara con otras tendencias arquitectónicas modernas como los contenedores y PaaS (Roberts, 2018).

  Entonces **FaaS reemplaza el servidor** (posiblemente una máquina física, pero definitivamente una aplicación específica) con algo que no necesita un servidor aprovisionado, ni una aplicación que esté ejecutando todo el tiempo, como es el caso de la identificación de billetes, esta función se ejecutará cuando el usuario lo requiera.

- *Con Lambda, se puede ejecutar código para casi cualquier tipo de aplicación o servicio backend sin tener que realizar tareas de administración.*

  Las ofertas de FaaS no requieren codificación a un marco específico o biblioteca. Las funciones de FaaS son aplicaciones normales cuando se trata de lenguaje y entorno. Por ejemplo, las funciones de AWS Lambda se pueden implementar en "primera clase" en Javascript, Python, Go, cualquier lenguaje JVM (Java, Clojure, Scala, etc.) o cualquier lenguaje .NET (Roberts, 2018). Sin embargo, AWS Lambda en realidad puede usar cualquier lenguaje que pueda compilarse en un proceso de Unix.

  Al desarrollar un código, lo único que se debe cambiar en un Faas es el "método principal" (main), ya que se elimina porque es el código específico para el controlador de mensajes de nivel superior o en otras palabras es la "interfaz de escucha de mensajes", pero esto podría ser solo un cambio en la codificación del método. El resto del código no es diferente en un mundo FaaS.

- *Solo se tiene que cargar el código y Lambda se encargará de todo lo necesario para ejecutar*

  La implementación es muy diferente de los sistemas tradicionales, ya que no se tiene aplicaciones de servidor para ejecutar por uno mismo. En un entorno FaaS, se carga el código desarrollado al proveedor de FaaS, y el proveedor hace todo lo necesario para aprovisionar recursos, instanciar máquinas virtuales, administrar procesos, etc.

- *y escalar*

  El escalado horizontal es completamente automático, elástico y administrado por el proveedor. Si su sistema necesita procesar 100 solicitudes en paralelo, el proveedor se encargará de eso sin ninguna configuración adicional de su parte. Los "contenedores de cómputo" que ejecutan sus funciones son efímeros, y el proveedor de FaaS los crea

y destruye por pura necesidad de tiempo de ejecución. Lo que es más importante, con FaaS, el proveedor maneja todo el aprovisionamiento y asignación de recursos subyacentes: el usuario no requiere ningún clúster o gestión de VM (Roberts, 2018).

- *el código con alta disponibilidad. Se puede configurar el código para que se active automáticamente desde otros servicios de AWS*

  Las FaaS generalmente se desencadenan por tipos de eventos definidos por el proveedor. Con AWS Lambda, tales estímulos incluyen actualizaciones S3, tareas programadas y mensajes agregados a un bus (por ejemplo, Kinesis) (Roberts, 2018).

- ***o se puede llamarlo directamente desde cualquier aplicación web o móvil***

  La mayoría de los proveedores también permiten que se activen funciones como respuesta a solicitudes HTTP entrantes; en AWS, normalmente se habilita esto mediante el uso de una API Gateway (Roberts, 2018).

2.3.1.1. Estado de las FaaS

Las funciones de FaaS tienen restricciones significativas cuando se trata del estado local (vinculado a máquina/instancia), es decir, datos que almacena en variables en la memoria o datos que escribe en el disco local. Se tiene dicho almacenamiento disponible, pero no se tiene garantía de que dicho estado persista en múltiples invocaciones y, más fuertemente, no se debe suponer que ese estado de una invocación de una función estará disponible para otra invocación de la misma función. Por lo tanto, las funciones de FaaS a menudo se describen como *sin estado*, pero es más preciso decir que cualquier estado de una función de FaaS que se requiera que sea persistente debe externalizarse fuera de la instancia de la función de FaaS.

2.3.1.2. Duración de la ejecución de una FaaS

Las funciones de FaaS generalmente están limitadas en cuanto tiempo, se permite ejecutar cada invocación. En la actualidad, el "tiempo de espera" para que una función AWS Lambda responda a un evento es como máximo cinco minutos, antes de finalizar.

2.3.2. API Gateways

Un aspecto de Serverless son las "API Gateways". Una API Gateway es un servidor HTTP donde las rutas y los puntos finales se definen en la configuración, y cada ruta está asociada con un recurso para manejar esa ruta. En una arquitectura sin servidor, tales controladores son a menudo funciones FaaS.

Cuando una API Gateway recibe una solicitud, encuentra que la configuración de enrutamiento coincide con la solicitud y, en el caso de una ruta respaldada por FaaS, llamará a la función FaaS relevante con una representación de la solicitud original. Por lo general, la puerta de enlace API permitirá la asignación de los parámetros de solicitud HTTP a una entrada más concisa para la función FaaS, o permitirá pasar toda la solicitud HTTP, generalmente como un objeto JSON. La función FaaS ejecutará su lógica y devolverá un resultado a la puerta de enlace API, lo que a su vez transformará este resultado en una respuesta HTTP que pasará de nuevo al llamante original (Roberts, 2018).

Un caso de uso para una API Gateway con funciones FaaS es la creación de microservicios HTTP de una manera Serverless con todos los beneficios de escalado, administración y otros que provienen de las funciones FaaS.

2.3.3. Ventajas y Desventajas

En la Tabla 2.1 se presenta un resumen de ventajas y desventajas al utilizar Serverless.

Ventajas	Desventajas
Escalamiento y administración de los recursos necesarios por parte del proveedor	Se restringe el acceso a máquinas virtuales, sistemas operativos o entornos de período de duración
Suministro ágil de los recursos en tiempo real, incluso en caso de picos de carga imprevisibles o un crecimiento desproporcionado	La implementación de las estructuras serverless requiere un gran esfuerzo
Solo se cobran gastos por recursos que realmente se han usado	Gran dependencia del proveedor ("efecto Lock-in"): en caso de cambiar de operador, por ejemplo, suele ser necesario escribir nuevamente todas las funciones basadas en eventos.
Gran tolerancia a errores gracias a la infraestructura flexible de hardware en los centros de cálculo del proveedor.	Procesos de monitorización y depuración de errores relativamente complejos, ya que, por norma general, no es posible realizar análisis profundos de errores y rendimiento.

Tabla 2.1: Ventajas y puntos débiles del concepto serverless.

2.4. Inteligencia Artificial y Aprendizaje Profundo

La inteligencia artificial se puede definir como la disciplina del campo de la informática que busca crear máquinas que permitan imitar comportamientos inteligentes. Dentro de este gran abanico de posibilidades caben tareas como *conducir un coche, predecir un valor numérico a largo plazo,* ***clasificar objetos*** *o generar contenido multimedia, entre otros.*

Las máquinas se programan para realizar estas acciones, llegando incluso a superar al ser humano en multitud de ellas, lo cual las convierte en herramientas totalmente funcionales y eficaces. Sin embargo, actualmente estas inteligencias artificiales no permiten consolidar una inteligencia holística como la que puede desarrollar la mente humana que sí puede realizar numerosas tareas de forma inteligente. Es decir, cada una de estas máquinas inteligentes están diseñadas para la realización de un fin concreto exclusivamente imitando comportamientos inteligentes (Martín, 2019).

Para acometer todos este tipo de acciones de forma inteligente por parte de las máquinas, es esencial que ésta tenga la capacidad de aprender. Para ello, nace el **aprendizaje automático**, que se puede definir como una rama de la inteligencia artificial que busca dotar a las máquinas de capacidad de aprendizaje. Esto se consigue mediante la generalización del conocimiento a partir de un conjunto de experiencias por las que la máquina aprende cómo realizar una acción y no solo a ejecutarla de forma automática.

Dentro de este campo, existen numerosas técnicas usadas para diversos fines como *la regresión lineal, la regresión logística, los árboles de decisión y sus variantes, la clusterización,* ***las redes neuronales****, etc.* Además, debido a la tendencia de estos modelos basados en redes neuronales en aumentar la complejidad añadiendo más y más capas, se genera el concepto de aprendizaje profundo o **deep learning** para aglutinar este tipo de algoritmos.

2.4.1. Redes Neuronales

Para obtener un modelo que servirá para realizar clasificación se debe entrenar al sistema, es decir, encontrar la relación o los patrones en los datos de entrada y la salida. Una de las mejores formas para reconocer imágenes es con redes neuronales (Neural Networks, NN), según Matich (2001) existen varias definiciones de qué es una red neuronal:

- Un modelo matemático compuesto por un gran número de elementos procesales organizados en niveles.
- Redes neuronales artificiales son redes interconectadas masivamente en paralelo de elementos simples (usualmente adaptativos) y con organización jerárquica, las cuales intentan interactuar con los objetos del mundo real del mismo modo que lo hace el sistema nervioso biológico.

Lo más interesante de las NN es su capacidad de aprender sin tener que enseñarles o indicarles cómo aprender o en qué fijarse para aprender. Lo que se necesita es tener un conjunto de fotos etiquetadas para que la NN sea capaz de aprender a reconocerlos. Por ejemplo, si se entrena un clasificador de fotos de billetes de 5 y 10 euros, la NN será capaz de determinar qué probabilidad existe de que esa foto sea de 5 euro o 10 euros.

Las NN son un modelo del que su desarrollo se ha visto afectado por las limitaciones de la tecnología de la época, pero aún así en las últimas décadas ha demostrado tener un gran potencial en muchos campos distintos. En el reconocimiento de billetes su potencial ha causado que diversidad de modelos surjan, principalmente basándose en la interpretación del billete como un conjunto de patrones.

Los procesos automatizados junto con la inteligencia adaptados a ellos para identificar billetes, surge por la necesidad de ejecutar esta tarea con ligereza. Cada billete tiene su complejidad debido a las distintas figuras plasmadas en ellos. Actualmente, cada billete tiene su propio "rostro" (Salcedo, 2017), que difiere claramente de aquellos de distinto valor aunque misma moneda. Esta complejidad es a su vez un modo de autenticidad y un método de clasificación; la imagen impresa sobre ellos es reconocible por las personas quienes pueden decir su valor.

Por lo tanto, para lograr el reconocimiento de cualquier billete, se debe determinar un método el cual permita equipar la percepción visual del ojo, y al mismo tiempo tener la rapidez y potencia de un ordenador. Todo esto, se consigue con la implementación de un modelo basado en redes neuronal ya que aporta ambas características.

Salcedo (2017) en su trabajo presenta el diseño de un algoritmo soportado en redes neuronales para la identificación de billetes mediante el reconocimiento de patrones los cuales obtiene al realizar un procesamiento de las imágenes. En este proyecto los billetes puestos a prueba son únicamente euros.

Otro proyecto con características similares es el realizado por Moretti et al. (2017), los cuales desarrollan un software para distinguir billetes argentinos en diferentes posiciones, escalas y incluso en forma parcial. Para este proyecto se han utilizado OpenCv y técnicas como "feacture detection", "descriptors matching", algoritmos como "homography", los cuales permiten que la aplicación tenga su propia inteligencia.

El trabajo realizado por Tamayo (2018) presenta un sistema de reconocimiento de billetes colombianos mediante la utilización de las técnicas que ofrece la visión artificial con OpenCv para el procesamiento de imágenes y la inteligencia artificial desarrollada con TensorFlow y Keras para la construcción de redes neuronales profundas.

En este trabajo se ha desarrollado un modelo de inteligencia artificial con redes neuronales. Este modelo de clasificación de billetes se desarrolla con la biblioteca **fast.ai** (Fast.ai,

2020) que permite el diseño de redes neuronales de aprendizaje profundo ayudado de librerías de visión para el procesamiento de las imágenes. Fast.ai está construida sobre PyTorch el cual es un nuevo marco de desarrollo para algoritmos inteligentes, simplificando la construcción y rápido despliegue de redes neuronales profundas.

2.4.2. Desarrollo de Modelos de Clasificación

Para el desarrollo de los modelos para algún fin específico, en este caso la clasificación de billetes, se encuentran disponibles varios marcos de desarrollo, dos bastante reconocidos son TensoFlow (2020) y Pytorch (2020).

Tanto TensorFlow como PyTorch son marcos de aprendizaje automático diseñados específicamente para desarrollar algoritmos de aprendizaje profundo con acceso a la potencia computacional necesaria para procesar muchos datos (por ejemplo, computación paralela, capacitación en GPU, etc).

Google lanzó en 2015 bajo la licencia Apache 2.0 lo que se conoce como TensorFlow y en octubre de 2019, lanzó TensorFlow 2.0, que se dice que es una gran mejora y cabe destacar que por lo general, se programa en Python. PyTorch, por otro lado, sale de Facebook y fue lanzado en 2016 bajo una licencia de código abierto igualmente permisiva. Como su nombre lo indica, también es una biblioteca de Python.

TensorFlow es "Definir y Ejecutar", mientras que PyTorch es "Definir por Ejecutar". En el marco Definir y ejecutar, se definen condiciones e iteraciones en la estructura del gráfico y luego se ejecutan. En *definir por ejecutar, la estructura del gráfico se define sobre la marcha durante el cálculo directo,* siendo una forma más natural de codificación.

Lo interesante con PyTorch y que se presenta de manera inmediata al realizar su búsqueda, es la conexión con sistemas Cloud como AWS, Google, Microsoft e incluso Alibaba.

2.5. Arquitectura Orientada a Microservicios

El término "Arquitectura de microservicios" ha surgido en los últimos años para describir un enfoque para desarrollar una sola aplicación como un conjunto de pequeños servicios, cada uno de los cuales se ejecuta en su propio proceso y se comunica con mecanismos ligeros, a menudo una API de recursos HTTP (Fowler y Lewis, 2014). Estos servicios se basan en las capacidades comerciales y se pueden implementar de forma independiente mediante maquinaria de implementación totalmente automatizada. Existe un mínimo de gestión centralizada de estos servicios, que pueden escribirse en diferentes lenguajes de programación y utilizar diferentes tecnologías de almacenamiento de datos.

La arquitectura orientada a microservicios es implementada para el desarrollo de la aplicación ya que permite tener por separado las funciones requeridas para su funcionamiento completo, por ejemplo la identificación de billetes se implementa con AWS Lambda y para el cambio de divisa se utiliza un servicio de terceros de internet que permite conocer el valor actual de euro o dólar. Estos dos servicios son ejecutados por separado, pero invocados dentro de la misma aplicación.

2.5.1. Comparación de Arquitectura Monolítica vs Arquitectura de Microservicios

Para comenzar a explicar el estilo de microservicio, es útil compararlo con el estilo monolítico: una aplicación monolítica construida como una sola unidad. Las aplicaciones empresariales a menudo se componen de tres partes principales: una interfaz de usuario del

lado del cliente (páginas HTML y JavaScript ejecutadas en un navegador del dispositivo del usuario), una base de datos y una aplicación del lado del servidor. La aplicación del lado del servidor manejará las solicitudes HTTP, ejecutará la lógica de dominio, recuperará y actualizará los datos de la base de datos, y seleccionará y completará las vistas HTML que se enviarán al navegador. Esta aplicación del lado del servidor es un monolito: un único ejecutable lógico. Cualquier cambio en el sistema implica construir e implementar una nueva versión de la aplicación del lado del servidor (Batra, 2020).

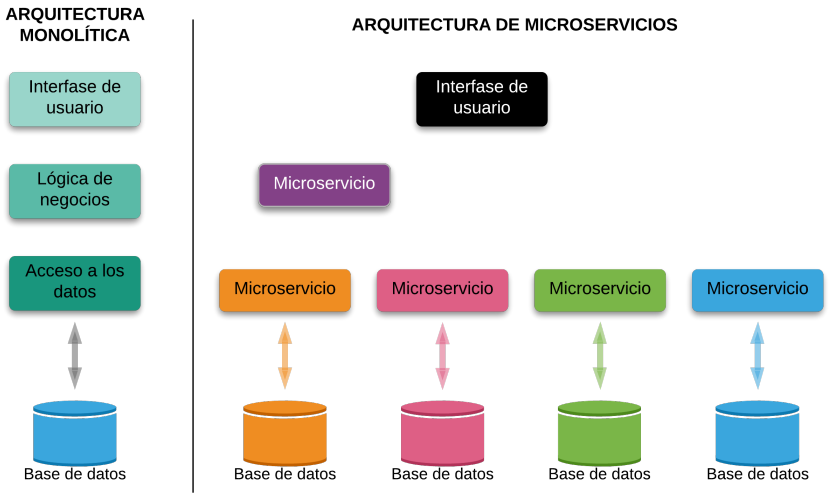

Figura 2.2: Arquitectura monolítica vs Arquitectura de Microservicios.

Tal servidor monolítico es una forma natural de abordar la construcción de dicho sistema. Toda su lógica para manejar una solicitud se ejecuta en un solo proceso, lo que le permite utilizar las características básicas de su idioma para dividir la aplicación en clases, funciones y espacios de nombres.

Las aplicaciones monolíticas pueden ser exitosas, pero cada vez más personas sienten frustraciones con ellas, especialmente a medida que se implementan más aplicaciones en la nube. Los ciclos de cambio están unidos: un cambio realizado en una pequeña parte de la aplicación requiere que todo el monolito sea reconstruido y desplegado. Con el tiempo, a menudo es difícil mantener una buena estructura modular, lo que dificulta mantener los cambios que solo deberían afectar a un módulo (Fowler y Lewis, 2014).

Estas frustraciones han llevado al estilo arquitectónico de microservicios: crear aplicaciones como conjuntos de servicios. Además del hecho de que los servicios son implementables y escalables de forma independiente, cada servicio también proporciona un límite de módulo firme, incluso permitiendo que se escriban diferentes servicios en diferentes lenguajes de programación. También pueden ser gestionados por diferentes equipos.

La inteligencia puede estar en el cliente o en el servidor, para este proyecto se ha definido que **la inteligencia de la identificación de billetes estará en la nube** puesto que los datos serán subidos y procesados allá, para finalmente bajar la decisión. La aplicación solo captura la fotografía y muestra la respuesta obtenida.

2.6. Aplicaciones móviles

En la sección 2.1.1 se presenta como dispositivo IoT a los teléfonos móviles por sus distintas capacidades. El apogeo de estos dispositivos cada vez aumenta, convirtiéndolos en una herramienta importante hoy por hoy especialmente para aquellos usuarios que poseen discapacidades sean estas de cualquier tipo, debido a que estos dispositivos ofrecen asistencia tecnológica para que puedan ser accedidos.

Con la aparición de la Web, los desarrolladores tuvieron que comenzar a crear aplicaciones para que puedan ser publicadas y accedidas, pero con la novedad que ahora los usuarios también lo pueden realizar también desde los dispositivos móviles (Ortiz et al., 2017), aprovechando las características de un desarrollo web para aplicaciones móviles.

2.6.1. Aplicaciones con accesibilidad

En esta sección se presentarán determinados puntos en común para el desarrollo de aplicaciones accesibles, encontrados en distintas fuentes como por ejemplo el estudio realizado por Ortiz et al. (2017), en el cual analizan guías existentes de usabilidad y accesibilidad para aplicaciones móviles, con el fin de obtener una guía unificada que ayuden al desarrollo de las aplicaciones.

También el Gobierno Español en su Portal de Administración Electrónica (Aguado y Estrada, 2017), brinda una guía de accesibilidad de aplicaciones móviles la cual puede ser utilizada por cualquier persona que tenga como objetivo la accesibilidad en el desarrollo de la aplicación. Esta guía presenta instrucciones de como generar este tipo de aplicaciones para los principales sistemas operativos del mercado (Android, iOS y/o Windows) y así subsanar posibles barreras que una aplicación puede presentar a usuarios con discapacidad.

Un blog con información interesante acerca de este tema es publicado por Estorach (2020). En él se presenta definiciones y diferencias entre usabilidad y accesibilidad, dos términos que suelen estar de la mano y no se deben confundir. Además, presenta características que poseen tanto los móviles con sistemas operativos iOS como Android para hacer uso de herramientas de accesibilidad, como por ejemplo en el caso de este último TalkBack el cual es un lector de pantalla incluido en el sistema operativo. Y el uso de otras aplicaciones como el Accessibility Scanner lanzado por Google, el cual te indica si la aplicación cumple o no estas características simplemente con abrirla.

Por último, Android (2020) presenta una serie de recursos que pueden ser utilizados como guía para el desarrollo, despliegue y pruebas de una aplicación accesible. Aquí se menciona que la información debe ser clara y específica, se debe tener una jerarquía en los objetos utilizados y agrupados de acuerdo a su relación con la información presentada. E igualmente menciona a Accessibility Scanner como una herramienta para evaluar si la aplicación cumple con características de accesibilidad.

Los punto en común en estas fuentes del desarrollo de aplicaciones móviles accesibles son los siguientes:

- En aspectos generales:
    - Las aplicaciones deben tener un tamaño pequeño que permita adecuarse a limitaciones técnicas de los dispositivos, aunque cada vez estos son más potentes.
    - Al utilizarse en dispositivos personales no requiere utilizar identificación para garantizar privacidad respecto a otros usuarios.
    - Existen características de accesibilidad propias del teléfono, estas funcionalidades deberían desaparecer del desarrollo de la aplicación para no interferir con

los servicios de accesibilidad nativos de la plataforma, dotando al usuario de una experiencia más consistente.

- En aspectos de diseño:

  - El foco del sistema es un elemento virtual que podemos asimilar como un puntero hacia un componente gráfico. En palabras simples, indica qué elemento de la pantalla está enfocado
  - El área útil de interacción debe ser de aproximadamente 9 mm de lado, pudiendo ser mayor si se desea. Del mismo modo, la separación entre las distintas áreas interactivas debería ser de al menos 1 mm para evitar la activación involuntaria de funciones (Aguado y Estrada, 2017).
  - Posibilidad de asociar los íconos con palabras. Esto sirve para que los programas que traducen a voz el texto, puedan leer la información de cada objeto.
  - Añadir texto descriptivo a los controles de la interfaz de usuario de la aplicación. El texto no debe describir visualmente el elemento al que representa, sino la acción asociada a dicho elemento. Es aconsejable que sea corto y conciso.
  - La asistencia por voz, se puede utilizar TalkBack el cual es una característica propia de Android y permite reproducir la información del objeto seleccionado.

2.6.2. Desarrollo de Aplicaciones Multiplataforma

Se conoce como aplicaciones multiplataforma a aquellas que pueden ejecutarse sin problemas en diferentes arquitecturas software y/o hardware (Ballesteros, 2019). La idea es *"Desarrolla una vez, despliega varias"*. Por ende, el desarrollar de tal forma que se pueda desplegar en varias plataformas permite economizar en esfuerzo y dar flexibilidad a los clientes para que usen su entorno preferido.

Los clientes de nuestros sistemas van a usar: móvil (Android, iOS), escritorio (Windows o Linux) o web (Chrome, Firefox, Electron). Tener un entorno que te permita ejecutar en varios es fundamental. Lo más versátil siempre es el web porque permite que todas las plataformas tienen soporte para navegadores. También es lo más restrictivo, o solía serlo.

Hay dos tipos principales de aplicaciones de este tipo, las que se escriben en un lenguaje interpretado (no compilado) como Java, o aquellas que se ejecutan en un servidor y se ofrecen a los clientes para que puedan acceder a ella. En este segundo grupo se incluyen las aplicaciones web. **Implementar una solución web para este problema es la mejor opción,** puesto que es la opción más sencilla y accesible para el usuario final, que podrá utilizarla específicamente en un móvil Android, pero también será adaptable a iOS y navegadores web.

Javascript no es Java, pero está dando lugar a variaciones muy relevantes en las que la instalación se minimiza. Aparte, los avances en Javascript dan lugar a opciones muy relevantes que han marcado el desarrollo en el lado del cliente desde hace ya varios años.

Otro factor a tener en cuenta al construir una aplicación es el uso de plataformas de desarrollo y más aún cuando se ha decidido que esta sea una solución web, *siendo **Apache Cordova** una de las más conocidas por su desarrollo enfocado en entornos web y desplegado en sistemas móviles.*

2.7. Lenguajes de Programación

Dentro del desarrollo de un proyecto y más aún en el ámbito de IoT, son muchos los lenguajes de programación que pueden encontrarse. De acuerdo a la encuesta publicada anualmente por Stack Overflow (2020), el cual es un sitio de preguntas y respuestas para programadores profesionales y aficionados, en 2020 se tiene a Javascript en primer lugar, seguido de HTML/CSS ya que van de la mano al realizar proyectos o aplicaciones con entornos web. En cuarto lugar se tiene a Python, el cual es bastante utilizado para proyectos de inteligencia artificial mediante diferentes frameworks como TensorFlow o PyTorch, y últimamente en servicios en la nube como AWS Lambda. Esto se observa en la Figura 2.3.

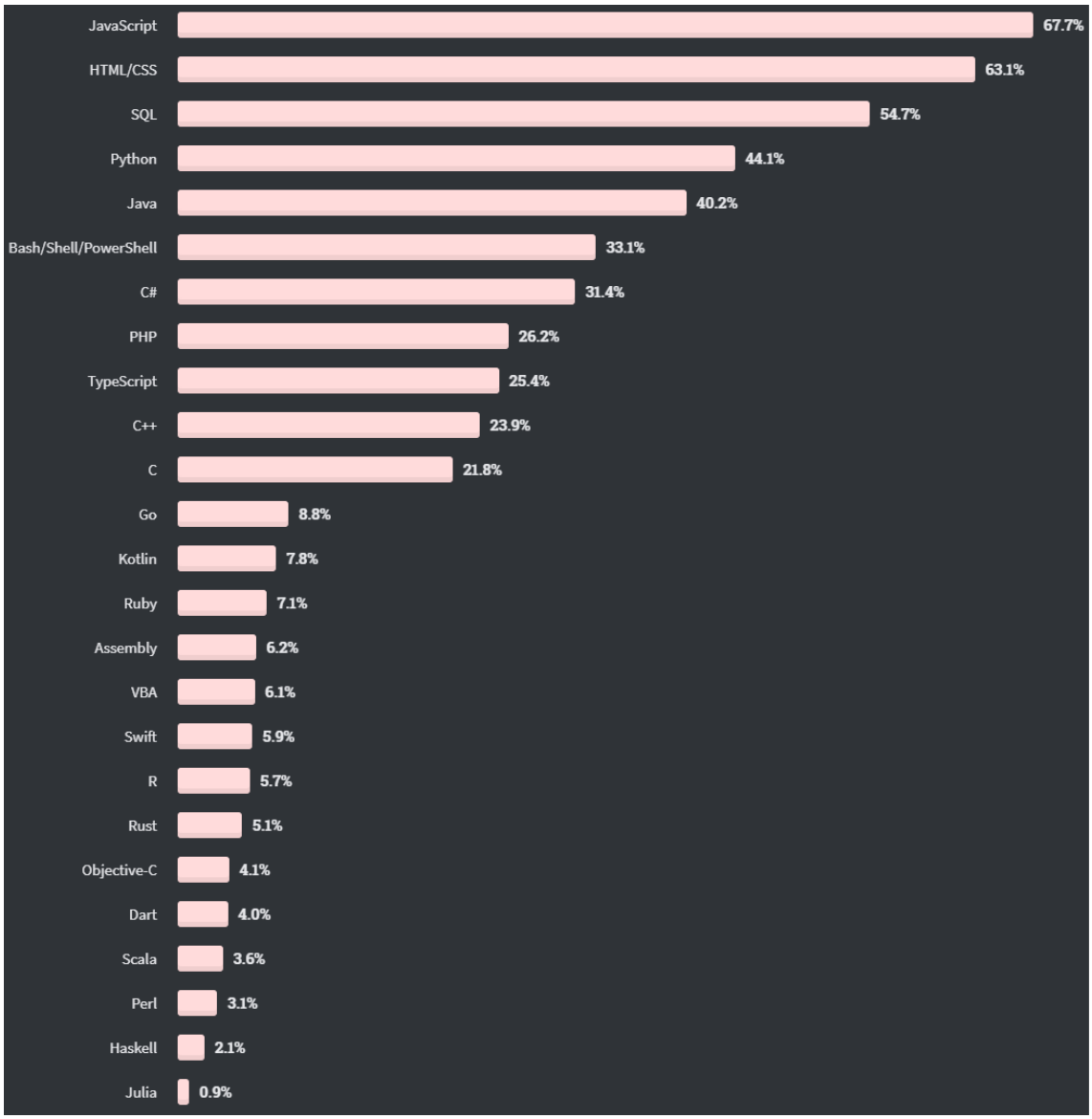

Figura 2.3: Lenguajes de programación más utilizados - 2020. Fuente: Overflow (2020)

2.8. Usuarios Finales: Personas con Discapacidad Visual

De acuerdo a lo publicado por la OMS, la deficiencia visual se clasifica en dos grupos de acuerdo al tipo de visión: de lejos y de cerca (OMS, 2018). Dentro del grupo de deficiencia de la visión de lejos, está la ceguera, que es una característica con la cual las personas carecen de visión total o no logran distinguir entre la obscuridad y la luz; impidiendo así la adquisición de conocimiento a través de la visión (Arteaga y Elizalde, 2007).

2.8.1. Desafíos para las personas con discapacidad visual

Las personas con discapacidad visual a menudo se encuentran con desafíos sociales y tecnológicos. Los sociales están relacionados con tareas que dificultan la participación de personas invidentes, frecuentemente la ceguera impide que estas personas realicen varios trabajos, limitando de esta manera las oportunidades de empleo, por lo cual afecta al desarrollo económico de la persona y su autoestima. Además la ceguera causa inconvenientes en la participación de actividades fuera del ámbito laboral, como son las académicas y deportivas. Muchos de estos desafíos sociales limitan la capacidad de conocer a otras personas, contribuyendo de esta manera a la baja autoestima (WAB, 2012).

Cuando las personas con discapacidad visual tienen que interactuar con la tecnología se presentan los otros desafíos, como por ejemplo leer información de una página web. Hoy en día existen plataformas de ayuda como los programas de asistencia de lectura que dan a conocer la información de páginas web a través de mensajes de audio, no obstante el aprendizaje de estos procesos requiere bastante tiempo.

2.8.2. Habilidades adquiridas para el reconocimiento de objetos

Ante estos desafíos las personas adquieren nuevas habilidades, en el caso específico de reconocer objetos la ONCE ha recalcado algunas de ellas y son la distinción figura-fondo, la constancia de la forma, el cierre visual, la memoria visual y la coordinación óculo-manual.

Las habilidades mencionadas son puramente perceptivas, su uso efectivo requiere estrategias paralelas. Ante la duda de que un objeto sea un vaso o una taza, la confirmación consiste en buscar el detalle que diferencia a ambos, es decir, el asa. También necesitará utilizar las relaciones lógicas: por ejemplo, si se busca una farmacia en un tramo concreto de una calle, la localización de un estímulo luminoso grande, de color rojo o verde, será suficiente para saber dónde se encuentra, aunque la cruz no se haya visto nítidamente ni en su totalidad (ONCE, 2011).

Los billetes, son objetos que están en el día a día de las personas con discapacidad visual ya que lo ocupan para realizar transacciones y con técnicas de doblado separan sus billetes o mediante el tacto logran reconocer de que valor son, aunque algunas veces suele complicarse. Ante esta situación, se han desarrollado aplicaciones que pueden hacer más fáciles estas tareas y que se citarán en el siguiente apartado.

2.9. Proyectos Similares

Google, por ejemplo en 2017 lanzó Google Lens (Google, 2020), que es una aplicación móvil gratuita para iOS y Android, de reconocimiento de imagen capaz de reconocer qué hay en una imagen y dar información relevante sobre ella, buscando “lo que ves”. Simplemente con escanear o capturar lo que se enfoca puede realizar lo siguiente: si es texto lo traduce en tiempo real, si es un lugar turístico lo identifica, muestra información sobre el

mismo y que hay alrededor, si es ropa sugiere lugares de compra en línea, si son animales muestra opciones de información (raza, origen, etc.) y en caso de dinero muestra noticias e imágenes sobre este tema, pero no describe de manera concreta la cantidad o denominación de los billetes o monedas enfocadas.

Microsoft por su parte desarrolló Seeing AI (Microsoft, 2020), que es una aplicación de inteligencia artificial, pero disponible solo para dispositivos iOS de manera gratuita. La aplicación utiliza la cámara para identificar personas y objetos, describiendo de manera audible esos objetos para personas con discapacidad visual.

Seeing AI es una aplicación únicamente en idioma inglés, que reconoce 5 divisas, Dólar estadounidense y canadiense, Libra esterlina, Rupia india y euro. El tamaño de la aplicación es de 282 MB, algo relativamente pesado. En pruebas realizadas cuando se abre hay que ir al selector de canales y activar la opción Currency (Divisas) para la asistencia en detección de billetes, por otra parte la aplicación en sí consume mucha energía y es un proyecto exclusivo de Microsoft, lo que no permite implementar nuevas características.

Otra aplicación es Cash Reader (Hayaku, 2019), desarrollada por Tomas Jelinek (desarrollador checo FreeLance, cursó sus estudios en informática en la Facultad de Tecnología de la Información de Brno, especializándose en Gráficos por computadora y Multimedia) que convierte los dispositivos móviles en una herramienta para el reconocimiento de billetes, obteniendo y facilitando la información del valor de los mismos de manera inmediata y precisa. Cash Reader reconoce el mayor número de monedas, de Australia a Latinoamérica. Creada pensando en la accesibilidad ya que se apunta a un billete y se oirá su denominación en tiempo real.

También se debe mencionar que si se pone la aplicación en silencio y se enfocan los billetes se leerán mediante vibraciones. Ideal para un lugar ruidoso o si se busca privacidad. Por ejemplo, 5 € sería una vibración. 10 € dos vibraciones, 20 € tres vibraciones, etc. La aplicación está desarrollada para dispositivos móviles iOS y Android, y tiene un costo de 15 € (Hayaku, 2019). Cabe mencionar que la aplicación tarda bastante en reconocer un billete, además requiere de una buena iluminación para hacerlo y que el billete debe estar extendido sobre una superficie plana y en horizontal, en caso contrario, normalmente no lo reconoce. Y por último, esta aplicación no se actualiza desde hace 15 meses.

NantMobile Money Reader, antes conocida como LookTel, es otra aplicación desarrollada para dispositivos iOS, se se puede abrir pulsando sobre el icono del menú o diciéndole a Siri Abrir «nan moni rider». La aplicación se abre y está lista para reconocer distintos tipos de divisas, como son: dólar estadounidense, canadiense y australiano, Dinar bahreiní, Real brasileño, Rublo bielorruso, Libra esterlina, euro, Shekel israelí, Rupia india, Yen japonés, Dinar kuwaití, Peso mexicano, Florín Húngaro, Dólar neozelandés, Zloty polaco, Rublo ruso, Riyal de Arabia Saudita, Dólar de Singapur y Dírham de los Emiratos Árabes Unidos. Cabe mencionar que la aplicación tiene un costo de 13.99 € y no se actualiza desde Octubre de 2017 (LookTel, 2020). Además, su desarrollo está en versión beta o de prueba desde la fecha mencionada.

El asistente a ser desarrollado en este proyecto es una prueba de conceptos en el ámbito de IoT, la idea principal es reconocer imágenes, específicamente billetes por medio de inteligencia artificial. Para ello se diseña una infraestructura por módulos, integrando servicios de la nube en un asistente móvil con características de accesibilidad. Cada módulo será independiente, teniendo la posibilidad de realizar modificaciones sin afectar el funcionamiento global y en el asistente a futuro se podrá incluir nuevas características o servicios ya que quedará disponible como código abierto. Igualmente la red neuronal podrá ser mejorada ya que estará en el mismo repositorio, pudiendo entrenarla con un nuevo conjunto de imágenes para obtener mejores resultados o incluir el reconocimiento de nuevas divisas.

Capítulo 3

Infraestructura IoT

"Any sufficiently advanced technology is indistinguishable from magic."
— Arthur C Clarke

El desarrollo de este capítulo está enfocado en el diseño de la infraestructura IoT del proyecto. Primero se presenta un esquema general en el cual se identifican dos grupos, la nube y el cliente, en ellos están los servicios y la aplicación, respectivamente. Cada grupo emplea distintas plataformas, tecnologías, herramientas y recursos que son descritos posteriormente. Estos están respaldados por los conceptos revisados en el Capítulo 2, permitiendo así el desarrollo y posteriormente la implementación de la aplicación descrita en el Capítulo 4.

Por último, se presenta el flujo del proceso que tiene este proyecto, permitiendo así entender la interacción de cada uno de los componentes utilizados y el por qué del uso de cada uno en determinados momentos de la ejecución como por ejemplo el almacenamiento del modelo de identificación de divisas en Amazon S3 y su posterior uso en AWS Lambda.

3.1. Esquema de la Infraestructura

En la Figura 3.1 se observa el esquema general de los recursos utilizados para formar la infraestructura IoT de este proyecto.

Por un lado, está **la nube** que en sí son los recursos de la plataforma Amazon Web Services (AWS) como AWS IAM, Amazon S3, AWS Lambda y Amazon API Gateway. También está el servicio para cambio de divisa en tiempo real en Internet que es tomado de una fuente de datos abiertos (Väin, 2020), estos datos son basados en la información del Banco Central Europeo. Los servicios elegidos de AWS en su mayoría son gratuitos, cada uno tiene un límite para pasar a ser pagados por su uso. Mientras que el servicio de cambio de divisa es totalmente gratuito.

Por otro lado, está **el cliente** que es la aplicación utilizada por el usuario objetivo del proyecto que son personas con discapacidad visual o ciegas, sin descartar que cualquier otra persona también puede usarla. Esta aplicación es desarrollada en un entorno web con Apache Cordova a través de contenedores y específicamente para la plataforma Android. Cabe recalcar que en transcurso del desarrollo del proyecto, se implementó para navegadores web con el fin de comprobar la correcta ejecución de los microservicios, esto debido a la facilidad que presenta el navegador Chrome para observar distintos mensajes a través de su consola.

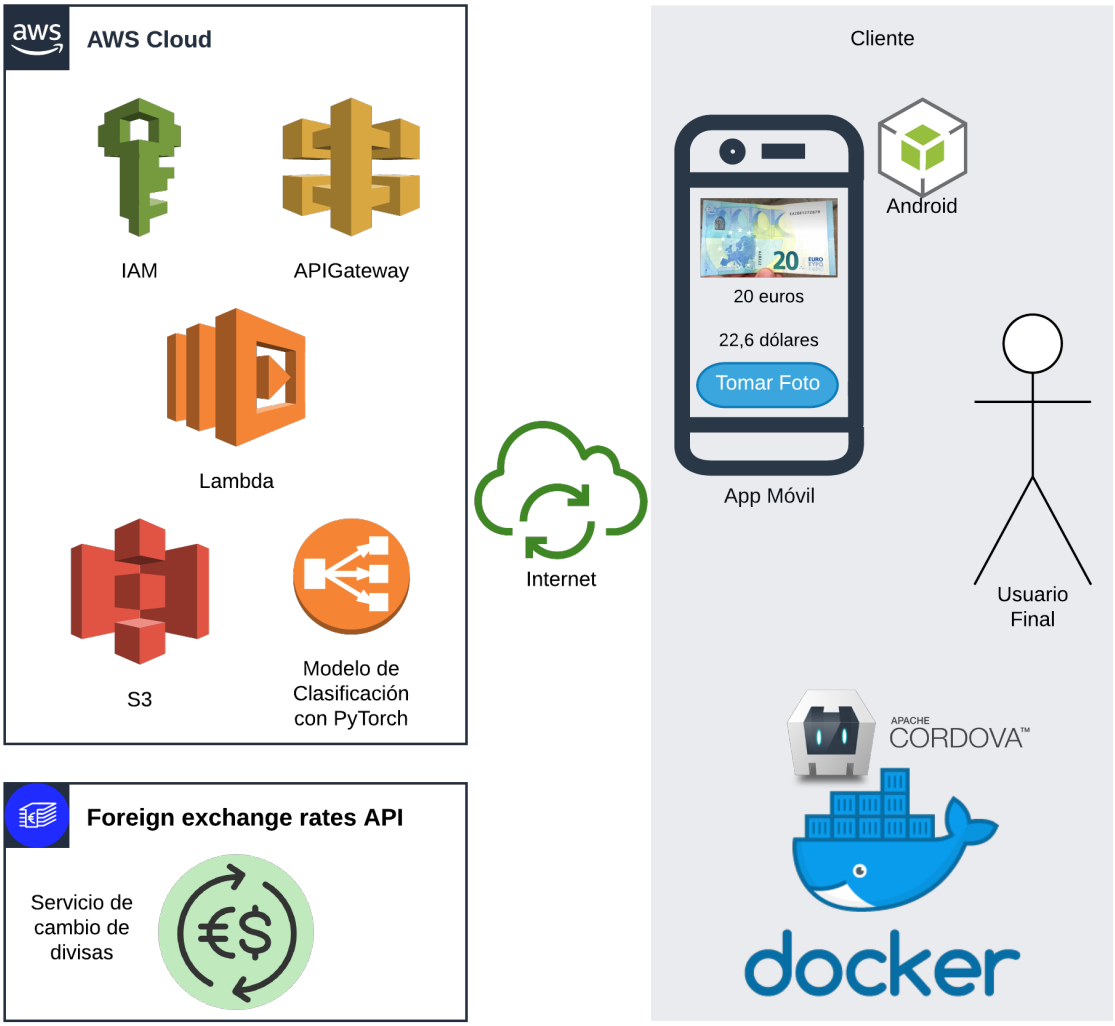

Figura 3.1: Esquema general de la infraestructura IoT del proyecto.

3.1.1. Servicios de la nube

La nube en este proyecto está conformada por la plataforma AWS que con ayuda de sus recursos permite desplegar un microservicio para identificación de billetes. Y del servicio Foreign exchange rates API el cual por medio de una API brinda información actualizada sobre el cambio de divisas actual en el instante de la consulta. A continuación, se describe cada recurso utilizado de manera breve y posteriormente se detalla cada uno de estos recursos con sus características.

- AWS IAM: Se utiliza para crear un usuario que permite ingresar tanto a la consola gráfica de AWS como trabajar desarrollos desde otros entornos como el local ya que brinda las credenciales necesarias para conectarse con AWS y sus servicios. AWS IAM es sencillo de usar y rápido de implementar. En este proyecto es requerido ya que el microservicio creado en Lambda se lo realiza en un entorno local y para poder llevar este desarrollo a la nube se necesitan las credenciales respectivas para conectar la cuenta creada en AWS con el entorno local.
- Amazon S3: Con este recurso **se crea un bucket el cual permite almacenar el modelo de clasificación de billetes** ya que una característica de la función

en Lambda es no utilizar archivos persistentes en su función como Serverless. Este recurso es utilizado porque tiene un almacenamiento gratuito hasta cierto límite, además que su conexión con Lambda es rápida y sencilla.

- AWS Lambda: Con este recurso de AWS **se desarrolla el microservicio de reconocimiento de billetes,** por medio de un código escrito en Python y utilizando herramientas como PyTorch ya que el modelo se acondiciona para este entorno de ejecución. Esta microservicio es implementado con el concepto de Serverless para que sea ejecutado al recibir una petición. Esto es una gran ventaja porque no es necesario tener un servicio ejecutándose siempre, únicamente bastará recibir una petición desde la aplicación para que se ejecute. Además, puede utilizarse por uno o varios clientes debido a su característica de escalabilidad y provisión de recursos que brinda AWS para la ejecución.

  Se debe tener en cuenta que la ejecución en frío afecta el funcionamiento, una característica descrita en la sección 2.3.1.1, por este motivo se ha implementado un disparo para "despertar" al microservicio cuando se abra la aplicación y las siguientes peticiones ya tengan una conexión estable con la función Lambda.

- Amazon API Gateway: **Permite crear un enlace para conectar las peticiones para reconocer los billetes por parte del cliente** y desde cualquier parte de Internet tener acceso al microservicio creado en AWS Lambda. Por medio de una API se indica el endpoint o recurso a ejecutarse. En otra palabras, brinda una interfaz para que la función pueda ser utilizada en este proyecto.

- El servicio de cambio de divisas es utilizado de una fuente de datos abiertos la cual proporciona esta información por medio de APIs. Para esto se deben hacer peticiones ajax desde el cliente solicitando información sobre el valor de las divisas en ese instante del euro a dólar.

  Foreign exchange rates API (servicio de cambio de divisas) es utilizado porque es gratuito, tiene una fuente de datos confiable como es el Banco Central Europeo y permite realizar consultas sin restringir a una cantidad límite.

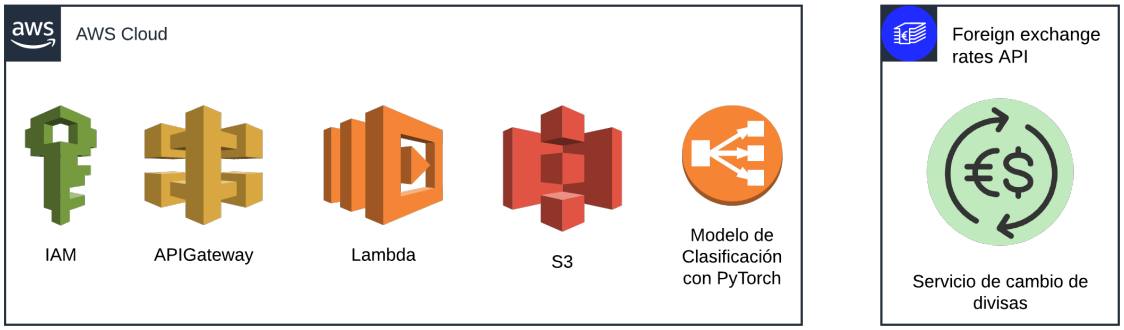

Figura 3.2: Esquema de la infraestructura IoT en la nube.

Se debe mencionar que se utiliza el modelo de aplicación Serverless de AWS (AWS SAM) que es un marco de código abierto para crear aplicaciones Serverless en AWS. Este tipo de aplicaciones son una combinación de funciones de Lambda, eventos y otros recursos que trabajan juntos para realizar tareas. Además, SAM permite incluir APIs para los desarrollos que se realicen.

Dentro de toda la documentación de AWS, SAM cuenta con varios templates que permiten un rápido despliegue mientras se aprende. Un ejemplo de como utilizar y desarrollar un Hello World se encuentra en AWS (2020c).

3.1.2. La aplicación como cliente

El cliente será la aplicación desde donde se ejecuten los microservicios en la nube, es decir la interfaz que tiene el usuario para interactuar. Para ello se muestra en la Figura 3.3 los recursos que se utilizan para el desarrollo de esta aplicación y que conforman el cliente como tal.

Figura 3.3: Esquema de la infraestructura IoT en el cliente.

- Docker: Es una **tecnología de contenedores la cual permite tener una plataforma de desarrollo, pero de manera portable que en este caso será Apache Cordova** junto con las dependencias que esta requiera. Esto ayuda a dejar de lado instalaciones y centrarse en el desarrollo de la aplicación.
- Apache Cordova: Es una **plataforma de desarrollo de aplicaciones multiplataforma y se la utiliza para crear la aplicación móvil**, en este caso se despliega para el sistema opertivo Android. Esto no significa que se desaproveche el concepto multiplataforma ya que para saber determinados funcionamientos como por ejemplo la conexión entre los microservicios, la aplicación puede ser desplegada en un navegador y utilizar la consola para observar las distintas conexiones. Aquí es donde se aprovecha el concepto "Desarrolla una vez, despliega varias". Se debe mencionar que Apache Cordova se obtendrá de una imagen dentro de Docker Hub, luego se procederá a montarla con Docker y ejecutarla para que permita el desarrollo de la aplicación.
- HTML5, CC3 y JS: son los lenguajes de programación utilizados para el desarrollo de la aplicación. Apache Cordova maneja la idea de entornos web, por este motivo estos lenguajes son los más adecuado debido a su frecuente uso en este tipo de aplicaciones. De acuerdo a la encuesta presentada anualmente por Stack Overflow (2020), estos lenguajes están dentro de los 3 primeros lenguajes más utilizados en el 2020.
- El cliente será un teléfono móvil con sistema operativo Android el cual permita instalación de servicios de terceros para poder llevar la aplicación a funcionamiento. Se deberá contar con una conexión a Internet y con una cámara como sensor.

- El usuario objetivo de este proyecto son personas con discapacidad visual o ciegas, sin descartar que cualquier otra persona también puede usarla. Por la situación actual de la pandemia del COVID-19, las pruebas se realizarán con diferentes personas (no se toma en cuenta la discapacidad).

3.2. Amazon Web Services

Amazon comenzó a expandir sus centros de datos para soportar la carga que suponía mantener su página web de comercio en línea. Posteriormente comenzó a vender parte de esa capacidad a sus socios y en 2006 abrió el servicio de AWS al público (AWS, 2020d). Existen otras compañías que ofrecen servicios en la nube similares, como Google o Microsoft, pero lo particular de AWS es que su interfaz se ha hecho tan intuitiva que no requiere conocimientos informáticos para comenzar a funcionar. Mediante dicha interfaz de usuario se puede empezar a operar en varios clics.

Figura 3.4: Servicios en la nube. Fuente: Barr (2020)

Para la consultora Gartner, AWS es considerado el líder para el desarrollo de servicios inteligentes en la nube debido a su capacidad de ejecución y visión para ofertar innovadores servicios como se observa en la Figura 3.4 (Barr, 2020), por detrás de AWS viene Microsoft y Google. Esto se debe a que AWS es realmente intuitivo de usar, permite el despliegue de cualquier servicio a través de APIs y la mayor parte de sus recursos cuenta con la

inteligencia para realizar tareas de escalado automático, como es el caso de AWS Lambda en el uso de este proyecto.

Gracias a AWS, cualquier persona o empresa podría crear una infraestructura desechable, en cuestión de minutos, por la que solo pagaría mientras utiliza y que podría eliminar después de forma completa o guardar una "referencia" a un coste muy reducido, con la que, en unos minutos, dicha infraestructura podría volver a estar funcionando en el mismo estado en que se dejó.

En la infraestructura IoT de este proyecto, AWS una plataforma de la nube en la cual se ejecutan tareas como almacenamiento (S3), función de inteligencia artificial para clasificación de billetes (Lambda) y creación de APIs para hacer uso de los recursos creados (API Gateway). Todos estos recursos son relevantes porque al hacerlos trabajar en conjunto permiten crear el microservicio requerido por la aplicación para identificar los billetes, el cual es un objetivo de este proyecto, por ello son descritos a continuación.

3.2.1. AWS IAM

Con AWS Identity and Access Management (IAM) se puede administrar el acceso a los servicios y recursos de AWS de manera segura. Además, se puede crear y administrar usuarios y grupos de AWS, así como utilizar permisos para conceder o negar el acceso de estos a los recursos de AWS. IAM es una característica de AWS que se ofrece sin cargos adicionales (AWS, 2020e).

Este servicio de AWS es utilizado porque permite tener seguridad (a nivel de usuario) para conectarnos a la nube y poder subir el desarrollo realizado en local de la función Lambda posteriormente descrita.

3.2.2. Amazon S3

Amazon Simple Storage Service (Amazon S3) es un servicio de almacenamiento de objetos que ofrece escalabilidad, disponibilidad de datos, seguridad y rendimiento líderes en el sector. Esto significa que clientes de todos los tamaños y sectores pueden utilizarlo para almacenar y proteger cualquier cantidad de datos para diversos casos de uso, como sitios web, aplicaciones móviles, procesos de copia de seguridad y restauración, operaciones de archivado, aplicaciones empresariales, dispositivos IoT y análisis de big data. Amazon S3 está diseñado para ofrecer una durabilidad del 99,999999999 % (11 nueves) y almacena datos de millones de aplicaciones para empresas de todo el mundo (AWS, 2020f).

Amazon S3 es utilizado por su característica de almacenamiento y disponibilidad de los datos. De esta manera otros recursos de AWS que no puedan tener información persistente, como el modelo de clasificación de billetes que se ocupará dentro de una función en AWS Lambda y más adelante se expondrá.

3.2.3. AWS Lambda

AWS Lambda permite ejecutar código sin aprovisionar ni administrar servidores. Se paga solo por el tiempo de cómputo que se consume.

Con Lambda, puede ejecutar código para casi cualquier tipo de aplicación o servicio backend sin tener que realizar tareas de administración. Solo tiene que cargar el código y Lambda se encargará de todo lo necesario para ejecutar y escalar el código con alta disponibilidad. Puede configurar su código para que se active automáticamente desde otros servicios de AWS o puede llamarlo directamente desde cualquier aplicación web o móvil (AWS, 2020b).

AWS Lambda dentro de la infraestructura IoT, ejecutará un microservicio que procesa la información enviada desde el dispositivo IoT y emite una respuesta, todo esto con ayuda de una API Gateway. Esta función ejecutada en Lambda se la hará con el concepto de Serverless Computing como FaaS que permite abstraerse de los recursos necesarios para ejecutar un código deseado ante determinados eventos como se observa en la Figura 3.5. Al no poder utilizar información persistente, se conecta con S3 para obtener el modelo que previamente deberá ser almacenado y así en cada momento de ejecución tomar este clasificador y ponerlo en funcionamiento.

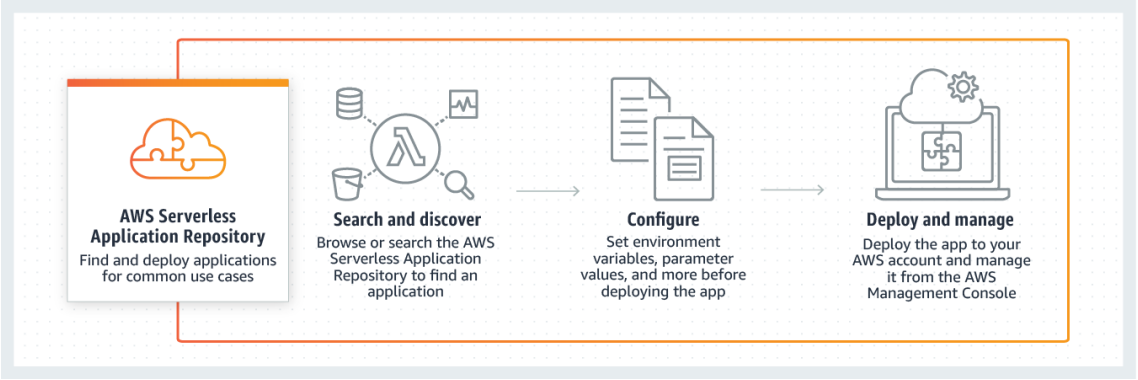

Figura 3.5: Funcionamiento de AWS Lambda como Serverless. Fuente: AWS (2020a)

3.2.4. Amazon API Gateway

Amazon API Gateway es un servicio completamente administrado que facilita a los desarrolladores la creación, la publicación, el mantenimiento, el monitoreo y la protección de API a cualquier escala. Las API actúan como la "puerta de entrada" para que las aplicaciones accedan a los datos, la lógica empresarial o la funcionalidad de sus servicios de backend. Con API Gateway, puede crear API RESTful y API WebSocket que permiten aplicaciones de comunicación bidireccional en tiempo real. API Gateway admite cargas de trabajo en contenedores y sin servidor, así como aplicaciones web (AWS, 2020g).

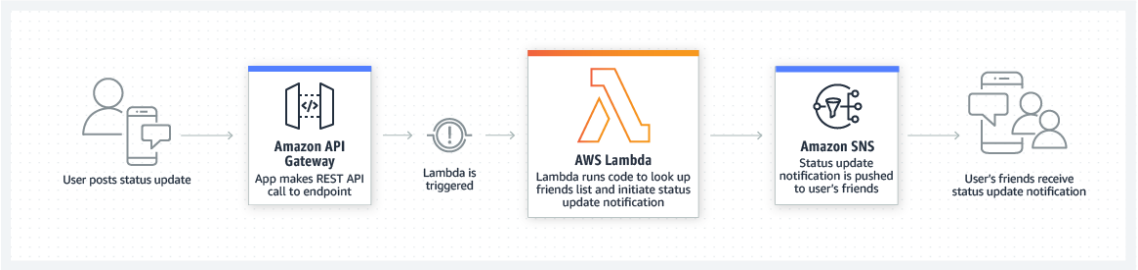

Figura 3.6: Ejemplo de Backend móvil para aplicaciones. Fuente: AWS (2020b)

API Gateway será un recurso fundamental ya que permite conectarse con la función de AWS Lambda por medio de una API Rest y brindar un microservicio al que la aplicación enviará una petición con la fotografía capturada y este microservicio dará una respuesta después de lo ejecutado con Lambda. Un ejemplo con características similares a este proyecto es mostrado en la Figura 3.6.

3.3. Servicio para Cambio de Divisas

Foreign exchange rates API with currency conversion es un servicio gratuito para los tipos de cambio actuales e históricos publicados por el Banco Central Europeo (Väin, 2020).

Este servicio es implementado en el proyecto debido a las siguientes características:

- **Open Source:** Todo, desde la base del código, la página de inicio hasta el proceso de implementación, es de código abierto y de uso gratuito bajo una licencia MIT permisiva.
- **Rápido y confiable:** Los tipos de cambio API se han diseñado y probado para manejar miles de solicitudes por segundo, los cuales son constantemente monitoreados.

La primera característica es fundamental ya que servicios como el cambio de divisas suelen ser restringidos o de pago. Además, su uso es bastante sencillo y ligero lo que permite implementarlo como un microservicio adicional de la aplicación. Y por último, esta respaldado con información de una entidad confiable como es el Banco Central Europeo.

3.4. Diseño del modelo de clasificación

Este proyecto tiene como premisa identificar billetes, para ello se debe contar con un modelo de clasificación el cual se diseñada siguiendo el flujo de trabajo mostrado en la Figura 3.7 que ayuda a dividir en pequeñas etapas todo el proceso hasta obtener el modelo.

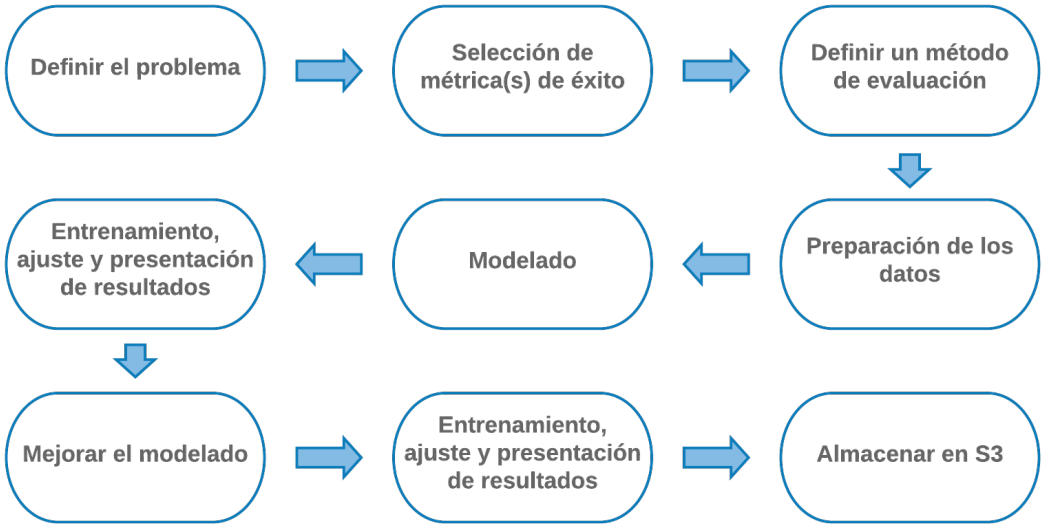

Figura 3.7: Flujo de trabajo para diseñar el modelo de clasificación.

Antes de continuar con la explicación del flujo de trabajo, cabe mencionar que este modelo está desarrollado con ayuda de la librería Fast.ai la cual se dedicada al desarrollo de inteligencia artifial y aprendizaje profundo con redes neuronales. Para ello esta librería ofrece modelos de redes neuronales pre entrenadas de manera general con distintas imágenes, pudiendo las personas tomar esta estructura de redes y entrenándolas con un fin

específico como es el caso de este proyecto, para identificar billetes. Y como se menciona en el estado del arte, esta librería utiliza el lenguaje de programación Python.

Continuando con el flujo de trabajo, en base a la Figura 3.7 se explica lo siguiente:

- En la primera etapa se *define el problema*, como se ha dicho a lo largo de este trabajo es la clasificación de billetes. Específicamente se identificarán dos tipos de billetes o divisas los cuales son euros y dólares, junto con el valor de cada uno, teniendo así distintas posibilidades como 1$, 10€ o 20 de ambas divisas.

  En resumen, es un problema de clasificación multiclase cuyo objetivo es identificar el tipo de billete y su valor, teniendo las siguientes combinaciones:

  - Tipo de billete: **euro**
    - Valores: **5, 10, 20, 50, 100, 200, 500**
  - Tipo de billete: **dólar**
    - Valores: **1, 5, 10, 20, 50, 100**

- El objetivo de la segunda etapa es *definir la métrica de éxito* para poder concluir si el modelo es aceptable o no. Para el diseño de este modelo las métricas de éxito son la **exactitud, precisión y recall**, estas últimas dos combinadas en una sola métrica (f1-score o fbeta), ya que son usadas frecuentemente en problemas de clasificación múltiple (Ng, 2020), permitiendo saber cuan efectivo es el modelo diseñado.

  Con ayuda de la Figura 3.8 se puede entender porque la exactitud está acompañada de la precisión y el recall. Una valor alto de precisión se consigue cuando los resultados están cercanos entre sí (no importa si están en circunferencias distintas), mientras que un valor exacto es aquel que se repite constantemente (siempre dentro de la misma circunferencia).

  Con esto se puede decir que:

  - la opción A, no tiene exactitud ni precisión. Tampoco recall.
  - la opción B, es precisa pero no exacta. Y su recall es del 50 %.
  - la opción C, no es precisa pero si exacta. Su recall es del 50 %.
  - la opción D, es precisa y exacta. Además, su recall es del 100 %.

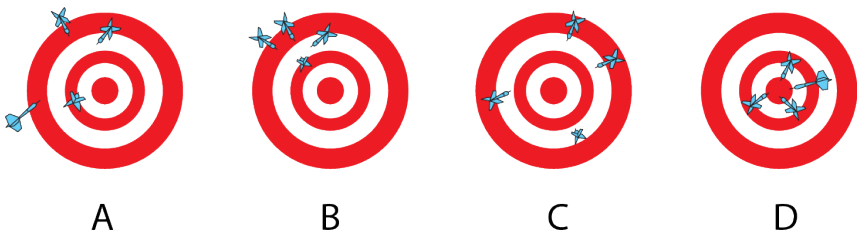

Figura 3.8: Comparación de las métricas de evaluación.

Estas métricas combinadas permiten saber si el modelo reconoce lo que realmente es, por ejemplo, si la fotografía tomada a un billete de 20$ es correctamente predicha como 20$ y no la confunde con un billete de 10$. En este último, estaría siendo preciso pero no exacto.

- En el tercer paso se define un método de evaluación. Esto se refiere a como el modelo se validará para presentar las anteriores métricas. Este modelo utilizará una validación cruzada, en el entrenamiento se tomarán pequeños grupos con los cuales se evalúa el modelo. Posteriormente, se validará este modelo con un grupo fuera de los datos de entrenamiento.

- La siguiente etapa es la preparación de los datos. Encontrar un conjunto o fuente de datos relacionado a este proyecto puede ser algo difícil, pero con la ayuda de Google Imágenes y fotos tomadas a varios billetes se ha logrado formar una fuente de datos.

  Esta fuente de datos consta con varias imágenes, tanto euros como dólares con los distintos valores presentados en la primera etapa. Estas imágenes han sido almacenadas en carpetas, con la estructura *tipo/valor* de billete. Teniendo un total de **440** distribuidos como se observa en la Figura 3.9.

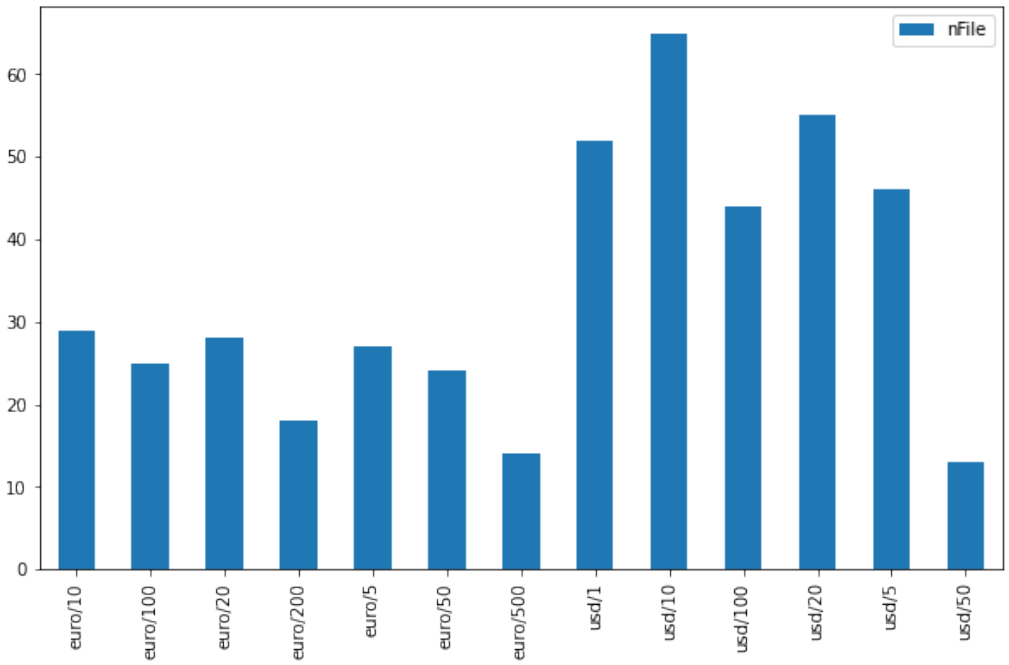

Figura 3.9: Cantidad de datos por clase.

- La quinta etapa esta dedicada para el modelado. El modelo inical es **resnet50**[1] el cual es tomado de los modelos proporcionados por la Librería Fast.ai y está construido para clasificación de imágenes con herramientas de PyThorch[2]. Este modelo es elegido por su enfoque de clasificar imágenes y por las características descritas en el artículo publicado por Howard y Gugger (2020), donde recalcan su rapidez en el entrenamiento con un conjunto de datos de ImageNet para obtener una precisión en 18 minutos, ganando así la competencia DawnBench de Stanford.

- La etapa seis consiste en realizar ajustes al modelado, variando sus parámetros como tasa de aprendizaje para disminuir el error y posteriormente entrenarlo con las imágenes de billetes. Con esto se obtienen resultados y en base a ellos decidir si puede o no mejorar la red.

[1] `https://docs.fast.ai/vision.models.html`
[2] `https://pytorch.org/docs/stable/torchvision/models.html`

- La séptima etapa se ejecuta en caso de que en el primer ajuste y entrenamiento, los primeros resultados no son tan óptimos, es decir, no se obtengan valores superiores al 90 % de las métricas de éxito.
- La octava etapa permite evaluar los resultados obtenidos del reajuste del modelo. Con esto determinar, sí con el reajuste el modelo logra una mejor clasificación de los billetes.
- La última etapa esta dedicada para guardar el modelo de acuerdo a los requerimientos para ser utilizada por la función Lambda. Esto se describe en la sección de implementación.

3.5. Docker como servicio de virtualización ligero

Docker es una plataforma abierta para desarrollar, enviar y ejecutar aplicaciones. Docker permite separar las aplicaciones de la infraestructura para que pueda entregar el software rápidamente (Docker, 2020a).

Docker es utilizado en este proyecto porque se aprovechan sus metodologías para enviar, probar e implementar código rápidamente, es decir se tendrá un contenedor con la plataforma para desarrollar la aplicación y el enfoque estará en escribir el código, construir la aplicación y ejecutarlo para su despliegue. No será necesario preocuparse por dependencias adicionales que suelen requerir las plataformas de desarrollo de aplicaciones.

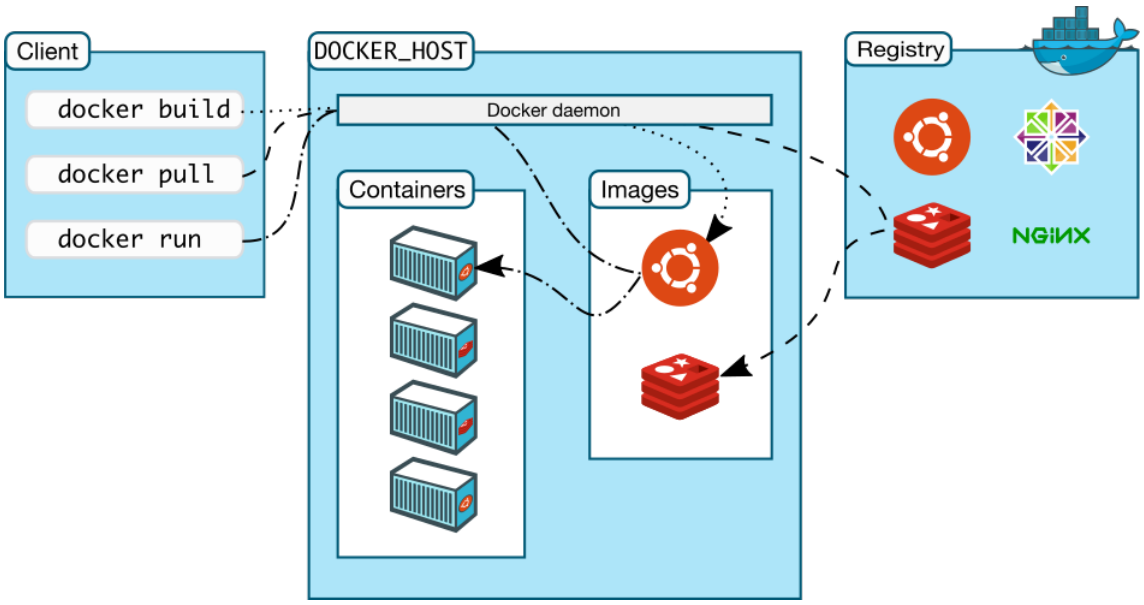

Figura 3.10: Arquitectura Docker. Fuente: Docker (2020a)

3.5.1. Docker Hub

Docker Hub es un servicio proporcionado por Docker para encontrar y compartir imágenes de contenedores con su equipo.

Proporciona las siguientes características principales (Docker, 2020b):

- Repositorios: imágenes de contenedores.

- Imágenes oficiales: permite extraer y usar imágenes de contenedores de alta calidad proporcionadas por Docker.
- Imágenes del editor: permite extraer y usar imágenes de contenedores de alta calidad proporcionadas por proveedores externos. Las imágenes certificadas también incluyen soporte y garantía de compatibilidad con Docker Enterprise.

Docker forma parte del desarrollo de la aplicación ya que Apache Cordova se utilizará como una plataforma portable dentro de un contenedor, evitando la preocupación de dependencias adicionales tanto en instalación como en despliegue de la aplicación.

3.6. Apache Cordova

Apache Cordova es un marco de desarrollo móvil de código abierto. Permite utilizar las tecnologías estándar web como HTML5, CSS3 y JavaScript *para desarrollo multiplataforma, evitando el lenguaje de desarrollo nativo cada plataformas móviles.* Aplicaciones ejecutan dentro de envolturas para cada plataforma y dependen de enlaces estándares API para acceder a de cada dispositivo sensores, datos y estado de la red (Cordova, 2020).

Con Apache Córdova se desarrolla la aplicación porque permite implentar los sensores incorporados al móvil y por medio de código javascript ejecutar los diferentes microservicios como identificar el billete y saber el valor de cambio de divisa de lo identificado. Además, su desarrollo basado en entornos web permite que la aplicación sea desplegable en varias plataformas, además de Android que es el caso de este proyecto.

Por lo general, Apache Cordova es utilizado por:

- desarrolladores móviles que desean extender una aplicación a más de una plataforma, sin tener que volver a implementarla con el conjunto de herramientas y lenguajes nativos de cada plataforma.
- desarrolladores web que desean implementar una aplicación web que esté empaquetada o embebida para la distribución en varias tiendas de aplicaciones.
- desarrolladores móviles interesados en mezclar componentes de aplicaciones nativas con un WebView (ventana de navegador especial) que puede acceder a las API a nivel de dispositivo, o si se desea desarrollar una interfaz de complemento entre componentes nativos y WebView.

Las plataformas que soporta Cordova para el desarrollo de aplicaciones son:

- Android,
- iOS, OS X (muy limitado),
- Windows Phone 8.1, Windows 8.1, 10
- Y por supuesto navegadores web.

Además, este marco de desarrollo móvil se lo puede encontrar en contenedores publicados en Docker Hub los cuales ya cuentan con todas las dependencias requeridas para el desarrollo inicial de una aplicación. Simplemente se debe tener Docker instalado, luego se busca en Docker Hub la imagen con la palabra clave “cordova”. Una vez identificada la imágen, se procede a implementarla con Docker como se describirá más adelante.

3.6.1. Arquitectura en Cordova

En una aplicación de Cordova existen varios componentes, en la Figura 3.11 se presenta una vista de alto nivel de la arquitectura en este marco de desarrollo móvil

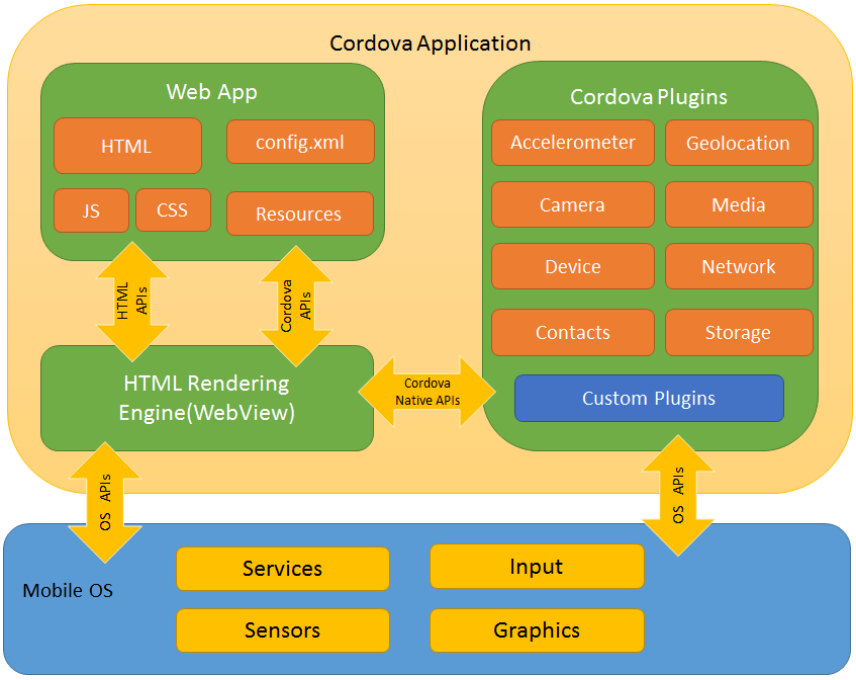

Figura 3.11: Arquitectura en Cordova. Fuente: Cordova (2020)

3.6.1.1. WebView

El WebView habilitado para Cordova puede proporcionar a la aplicación toda la interfaz de usuario. En algunas plataformas, también puede ser un componente dentro de una aplicación híbrida más grande que mezcla WebView con componentes de aplicaciones nativas. Las aplicaciones de Cordova normalmente se implementan como un WebView basado en navegador dentro de la plataforma móvil nativa Cordova (2020).

3.6.1.2. Web App

Esta es la parte donde reside el código de la aplicación, en sí se implementa como una página web, de manera predeterminada, un archivo local llamado index.html, que hace referencia a CSS, JavaScript, imágenes, archivos multimedia u otros recursos son necesarios para que se ejecute. La aplicación se ejecuta en un WebView dentro del contenedor de aplicaciones nativo, que distribuye a las distintas tiendas de aplicaciones (Cordova, 2020).

Este contenedor tiene un archivo muy crucial: el archivo config.xml que proporciona información sobre la aplicación y especifica los parámetros que afectan su funcionamiento, como si responde a los cambios de orientación.

3.6.1.3. Plugins

Los plugins (complementos) son una parte integral del ecosistema de Cordova ya que proporcionan una interfaz para que Cordova y los componentes nativos se comuniquen

entre sí y los enlaces a API de dispositivos estándar. *Esto permite invocar código nativo desde JavaScript* Cordova (2020).

El proyecto Apache Cordova mantiene un conjunto de complementos llamados Core Plugins. Estos complementos básicos proporcionan su aplicación para acceder a las capacidades del dispositivo, como la batería, la **cámara**, los contactos, etc. Además de los complementos principales, hay varios complementos de terceros que proporcionan enlaces adicionales a funciones que no necesariamente están disponibles en todas las plataformas.

3.7. Descripción del dispositivo IoT

Como anteriormente se describió en la sección 2.1.1, un teléfono móvil es considerado un dispositivo IoT. Para que la aplicación desarrollada en este proyecto pueda funcionar, el teléfono móvil debe tener los siguientes requerimientos mínimos:

- El sistema operativo sea Android
- Tenga cámara incorporada.
- Permita la instalación de programas de terceros.
- Permita conexión a internet.

Con estos requerimientos el dispositivo móvil podrá instalar y ejecutar la aplicación creada. Su funcionamiento es primordial ya que tiene la capacidad para procesar las distintas acciones de la aplicación:

- Tomar fotografías de billetes (euros y dólares de varias denominaciones).
- Enviar estas fotografías por internet hacia el microservicio creado en Lambda. Quedando a la espera de la respuesta para saber la identificación del billete y su valor.
- Luego envía otra solicitud pero esta vez hacia el servicio de cambio de divisas, para conocer la equivalencia actual.
- Posteriormente presenta la información obtenida por medio de la aplicación al usuario.
- Con herramientas de accesibilidad propias del teléfono, previamente activadas, la aplicación por medio de voz describirá la información presentada.

3.8. Diseño de la interfaz de la aplicación

Dentro de la infraestructura IoT está el teléfono móvil como dispositivo IoT presentado anteriormente. En él se instala y ejecuta la aplicación desarrollada para la identificación de billetes a ser utilizada por personas ciegas. Entonces la aplicación debe tener características de accesibilidad en su diseño y esto se logra con ayuda de los puntos en común y de las guías presentadas en el estado del arte.

3.8.1. Aspectos generales

Primero se toman en cuenta detalles generales para la aplicación. Uno de ellos es identificar los usuarios objetivos, este proyecto esta enfocado a personas ciegas. En base a los usuarios se determina que el mejor mecanismo para transmitir la información es por voz y Android cuenta con TalkBack descrito en la literatura como una característica de accesibilidad. Esto se deberá ajustar en el teléfono móvil.

TalkBack es utilizado ya que dentro de las guías presentadas para el diseño recalcan que de existir funciones de accesibilidad disponibles, es mejor hacer uso de ellas en cuanto a programación para no interferir con los servicios nativos.

Otro aspecto importante a tener en cuenta es el tamaño de los dispositivos móviles, el cual puede estar en un promedio de 4.5".

3.8.2. Aspectos de diseño

Para el diseño de la interfaz de la aplicación, la literatura menciona varias consideraciones, entre ellas es tener una estructura que permita separar funciones específicas a realizarse por la aplicación. Por esto, se ha decido dividir en 3 secciones, que se describen a continuación y se observan en la Figura 3.12.

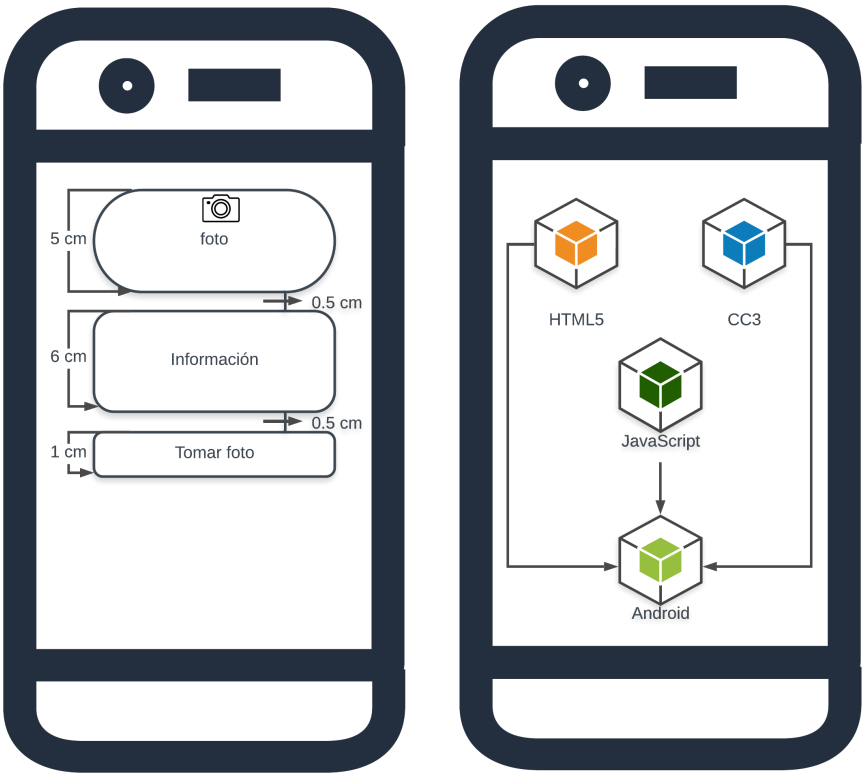

Figura 3.12: Esquema de la interfaz para la aplicación.

- La primera sección es un contenedor que permite mostrar la fotografía capturada. Esta sección tiene una dimensión de 5 cm de alto.
- La segunda sección está dedicada a presentar la información devuelta por parte de los servicios, valor y denominación del billete y su respectivo cambio de divisa. Esta

sección se ayudará de la característica TalkBack que permite reproducir a través de voz la información presentada. Tiene un alto de 6 cm.

- La tercer sección es un botón que accede a la cámara para tomar la fotografía del billete, después de realizar la fotografía, ejecuta todos los servicios en la nube. El botón tiene un dimensión de 1 cm de alto.

Además, las distancias de separación han sido de mayor dimensión (0.5cm) que las presentadas en el estado del arte (0.1cm). Esto se hace con el fin de facilitar el manejo de la aplicación ya que con el foco de desplazamiento y TalkBack activado, se podrá identificar los distintos componentes dentro de la aplicación.

En la Figura 3.12 se observa un espacio adicional, esto se debe a que la información presentada en algunos casos será mayor ya que se mostrarán las alternativas de identificación del billete debido a que el modelo no ha podido identificar una clase concreta.

Este diseño posteriormente será implementado con los lenguajes de programación que se observan en la parte derecha de Figura 3.12 ya que Apache Cordova permite plasmar un entorno web en una aplicación móvil.

3.9. Flujo de la información

En la Figura 3.13 se muestra como interactúa la aplicación con los diferentes microservicios desarrollados y tomados de la nube. Esto se describe a continuación.

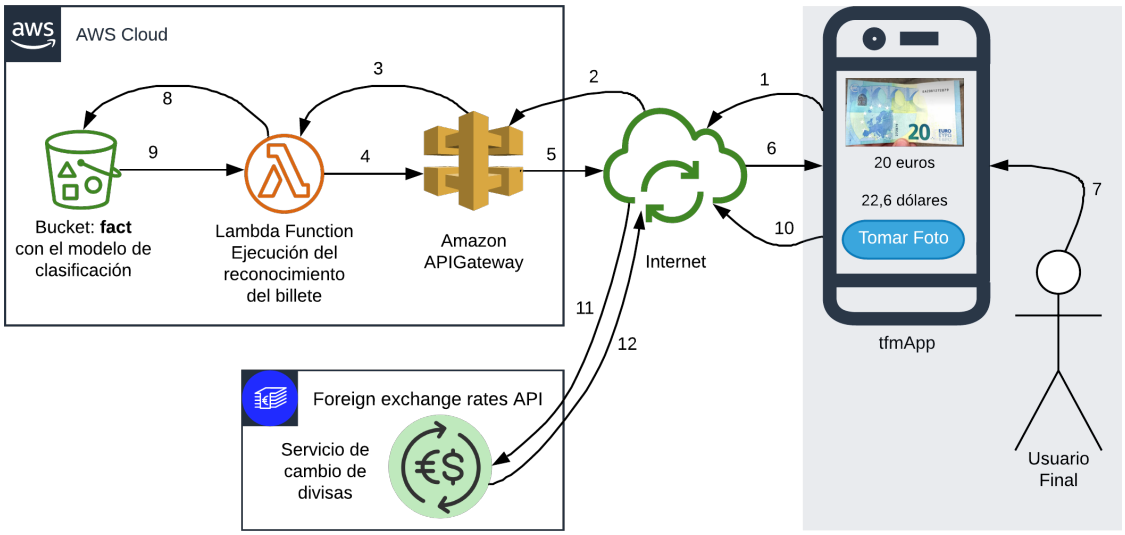

Figura 3.13: Flujo de la información en la infraestructura IoT del proyecto.

Ejecución en frío, es la primera interacción de la aplicación con un servicio en la nube la cual es Lambda que trabaja bajo peticiones, pero se pueden dar casos que no exista conexión o simplemente no se ejecute el código implementando y para ello se realiza esta comprobación.

1 Al abrir la aplicación, se envía un mensaje de "ping" hacia internet por medio de la API Gateway. En la literatura se recomienda evitar los arranques en frío de las funciones serverless ya que son funciones que no poseen estado, es decir su ejecución no se tiene la garantía de que todas las variables retengan la información de invocaciones previas. Por este motivo se realiza una primera conexión con un ping.

2 Internet encuentra y conecta la aplicación con la dirección de la API Gateway.

3 La API ya tiene los recursos y métodos definidos, esto permite direccionar la información recibida hacia la función Lambda como Serverless que ejecutará el código.

4 Lambda en su función "lambda_handler" reacciona ante eventos, en este caso es una petición recibida. La información contiene un "ping" (primera petición de la aplicación) y ante ello se programa que responda con un "pong".

5 La API Gateway toma el mensaje pong y lo envía hacia internet.

6 Internet envía este mensaje al cliente que se conectó por medio de la aplicación. De esta manera se ha ejecutado la primera solicitud o petición, con el fin de evitar arranques en frío con información más importante o simplemente más grande como puede ser una imagen.

Reconocimiento de billetes, esto se ejecuta con la intervención del usuario después del arranque en frío.

7 El usuario por medio de su teléfono móvil como dispositivo IoT y su cámara como sensor, procede a tomar una fotografía. Al capturarse, la imagen es enviada inmediatamente.

Los pasos 1, 2 y 3 son ejecutados como en la ejecución en frío. Salvo que la información en este caso, es la fotografía tomada.

8 La función Lambda ejecuta su código ante esta nueva petición, esta vez de clasificación. Como Lambda no almacena información persistente, hace uso de S3 para obtener el modelo de clasifición.

9 En S3, se tiene el bucket **tfm-iot-fact** en el cual está el modelo de clasificación requerido por Lambda.

Una vez recibido el modelo, se realiza el procesamiento de clasificación dentro de Lambda y se emite una respuesta. Luego se repiten los pasos 4, 5 y 6, mostrando en pantalla el resultado obtenido.

10 Al obtener el valor y denominación del billete, se ejecuta un segundo microservicio que es el cambio de divisas. Para ello la aplicación se conecta por medio de una API al servicio Foreign exchange rates API para solicitar la información del cambio en ese instánte de EURO a DOLAR.

11 Internet conecta la aplicación con esta API.

12 La API envía a Internet un JSON con la respuesta ante la solicitud enviada, que posteriormente es dirigda a la aplicación (paso 6). En la aplicación se recibe la información y se muestra el cambio de divisa.

Con esto se completa los ciclos de ejecución de la aplicación. En el teléfono móvil se debe tener activa la característica de accesibilidad nativa de Android como es TalkBack o Voice Assistant, esta herramienta permitirá que la aplicación por medio de voz informe tanto de la información presentada como de los elementos que tiene para realizar las distintas acciones.

Capítulo 4

Implementación

"Failure is simply the opportunity to begin again, this time more intelligently."
— Henry Ford

4.1. Requerimientos iniciales

Para el desarrollo de este trabajo se utilizaron los diferentes recursos mostrados en el capítulo 3. Para ello se ocupó una máquina virtual con sistema operativo Linux de escritorio, **Ubuntu 20.04 LTS** en la cual se realizó este proyecto.

Tanto para el desarrollo de la aplicación móvil como del microservicio en AWS Lambda se tuvo que preparar el entorno, esto se puede observar en:

- Apéndice A.1 - Preparación de la plataforma Apache Cordova con Docker
- Apéndice A.2 - Preparación de AWS SAM

4.2. Implementación de recursos y servicios en AWS

Para utilizar AWS, primero se debe tener una cuenta la cual puede ser una básica y para ello se debe registrar en `http://aws.amazon.com/`.

Una vez creada la cuenta se procede a ingresar a la consola de administración de AWS para configurar los recursos necesarios para el desarrollo del microservicio de reconocimiento de billetes en AWS Lambda.

4.2.1. Creación de Usuario en AWS IAM

Con AWS IAM se procede a crear un usuario y posteriormente sus credenciales con ayuda de la documenación de AWS mostrada en el siguiente enlace `https://docs.aws.amazon.com/es_es/IAM/latest/UserGuide/id_users_create.html`.

Es necesario la creación de este usuario junto con las credenciales para conectar la plataforma de AWS con el entorno local preparado debido a que se utilizará AWS SAM para el despliegue de la función Lambda como serverless para obtener el microservicio de reconocimiento de billetes.

4.2.2. Creación de Bucket en Amazon S3

Amazon S3 permite crear un espacio para almacenamiento de objetos en la nube denominado bucket el cual debe tener un nombre único en el entorno de AWS. Este bucket permite almacenar el modelo de clasificación ya que como se ha mencionado la función Lambda al trabajar como serverless, no almacena datos persistentes y por ello se ayuda de S3 para tener a disposición este tipo de archivos.

Para crear un bucker se utiliza la ayuda documentada[1] de AWS. En este proyecto algunos campos solicitados durante la creación del bucket han sido llenados con la siguiente información, el resto se ha mantenido lo indicado por la guía.

- Nombre del bucket: **tfm-bucket-fact**
- Región: **EU (Ireland)**
- Versioning: Solo se ha seleccionado *mantener todas las versiones del objeto*

Para poder subir u obtener archivos del bucket, se debe habilitar permisos y generar una política de acceso. En este caso se deja con acceso público para obtener el modelo de clasificación requerido en Lambda, esto se realiza con ayuda de la documentación[2] de AWS. En la Figura 4.1 se puede observar la política implementada para obtener o subir archivos al bucket creado y el acceso público configurado.

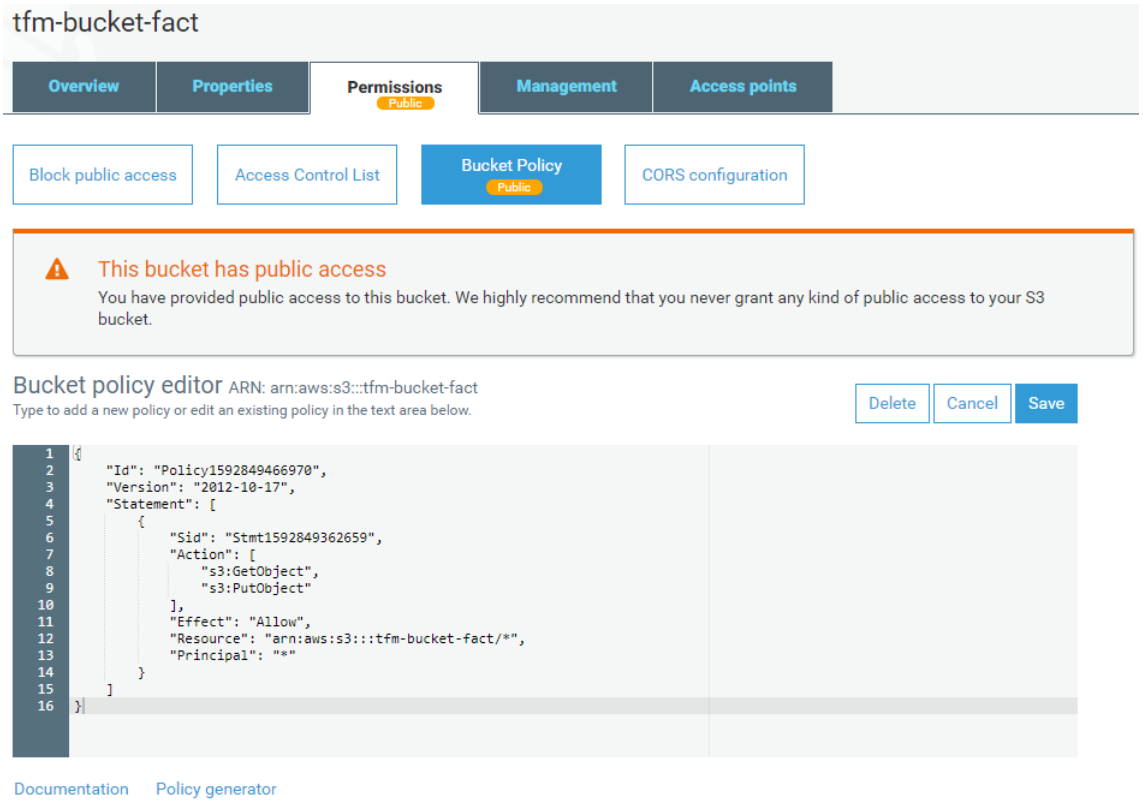

Figura 4.1: Implementación de política de seguridad en el bucket creado.

[1] https://docs.aws.amazon.com/es_es/AmazonS3/latest/gsg/CreatingABucket.html

[2] https://docs.aws.amazon.com/es_es/AmazonS3/latest/dev/access-control-block-public-access.html

4.3. Desarrollo del modelo de clasificación

A continuación, se presenta el desarrollo para obtener el modelo de clasificación de billetes en base al diseño presentado en la sección 3.4, en la cual de manera implícita se realizan las tres primeras etapas que son:

1 Identificación del problema multiclase para identificación de billetes,

2 Selección de medidas de éxito: exactitud, precisión y recall para evaluar el modelo

3 Y la selección del método de evaluación del modelado

En el Apéndice B.1 se presenta la implementación de todo lo descrito en las siguientes secciones.

4.3.1. Preparación de los datos

Primero se importan los datos de la fuente de almacenamiento, consiguiendo un listado de ítems los cuales son cada una de las imágenes que tienen como características: un nombre y ruta de almacenamiento.

Luego los datos son divididos en dos conjuntos: entrenamiento y validación, con 350 y 87 imágenes, respectivamente. Una vez obtenido estos conjuntos se procede a *codificar cada item (imagen)*. La codificación consiste en que cada entrada tenga una salida, es decir, cada imagen tendrá su respectiva identificación o etiqueta. Por ejemplo, la primera imagen del conjunto de datos es un billete de 20 euros y así sucesivamente para todas las imágenes de los conjuntos habrá una etiqueta.

Esto se logra con ayuda de la ruta de cada imagen ya que en ella se tiene el tipo y valor del billete que representa cada imagen, descrito en la sección 3.4 donde se crea la fuente de datos. En las Figuras 4.2 y 4.3 se observa la importación y codificación de las imágenes.

```
ItemLists;

Train: ImageList (350 items)
Image (3, 204, 246),Image (3, 225, 225)
Path: /content/drive/My Drive/imgs;

Valid: ImageList (87 items)
Image (3, 500, 281),Image (3, 500, 375)
Path: /content/drive/My Drive/imgs;
```

Figura 4.2: Datos importados sin codificar - Listado de Items.

```
LabelLists;

Train: LabelList (350 items)
x: ImageList
Image (3, 204, 246),Image (3, 225, 225)
y: MultiCategoryList
euro;10,euro;10,euro;10,euro;10,euro;10
Path: /content/drive/My Drive/imgs;

Valid: LabelList (87 items)
x: ImageList
Image (3, 500, 281),Image (3, 500, 375)
y: MultiCategoryList
usd;5,euro;20,usd;10,usd;100,usd;20
Path: /content/drive/My Drive/imgs;
```

Figura 4.3: Datos codificados - Listado de Etiquetas.

Siguiendo con la preparación de los datos y con ayuda la documentación de fast.ai, ya con los datos codificados se procede a formar el databunch (grupo de datos) de imágenes que es requerido para el entrenamiento y ajuste del modelo.

Este databunch permite definir un grupo de imágenes que se utilizan para la validación cruzada cuando se realiza el entrenamiento. Debido a la poca cantidad de datos, el grupo

de validación se forma en base a las propias imágenes, pero variando características como rotación, enfoque, iluminación, entre otras.

Además, con las herramientas de visión de la librería fast.ai se define un tamaño estándar de las imágenes, también se normaliza el conjunto de datos con los valores de la desviación media y estándar de las imágenes del conjunto de entrenamiento de ImageNet. ImageNet es la colección de 14 millones de imágenes utilizadas para entrenar previamente la arquitectura resnet50.

Entonces una vez listos los datos, se procede a realizar el modelo. Antes de continuar, se presenta las etiquetas o clases que tiene que clasificar el modelo, las cuales son:

"1", "10", "100", "20", "200", "5", "50", "500", "euro", "usd"

4.3.2. Modelado

En esta sección se presenta las etapas 5, 6, 7 y 8 del flujo de trabajo (Figura 3.7) que son el modelado, entrenamiento, ajuste, presentación de resultados y un reajuste del modelo para mejorar sus métricas de éxito.

Para crear el modelo se usa como base una red neuronal convolucional (CNN), específicamente **resnet50** proporcionada por fast.ai. Resnet50 es una red de varias capas diseñada especialmente para clasificar imágenes y también para trabajar en reconocimiento facial. Estas características se adaptan para obtener un modelo para la identificación de billetes, especialmente por los dólares que tienen rostros de personas en cada uno.

El modelo ha sido iniciado con la siguiente información, pasando como datos el databunch creado, la red a utilizarse y las métricas. La red también cuenta con la característica que los pesos de su última capa pueden ser utilizados para entrenamientos con fines específicos. Las métricas definidas son exactitud y el promedio (fbeta) entre la precisión y el recall, este último se debe ya que fast.ai no define las dos últimas métricas por separado.

```
model = cnn_learner(data, models.resnet50, metrics=[partial(accuracy_thresh,
thresh=0.2), partial(fbeta, thresh=0.2)])
```

La implementación fue con ayuda de las funciones proporcionadas por fast.ai, las cuales son ejecutas de la siguiente manera hasta conseguir un modelo óptimo.

- Como primer paso, se debe obtener una tasa de aprendizaje para el entrenamiento y para ello se utiliza `model.recorder.plot()` obteniendo lo mostrado en la Figura 4.4.

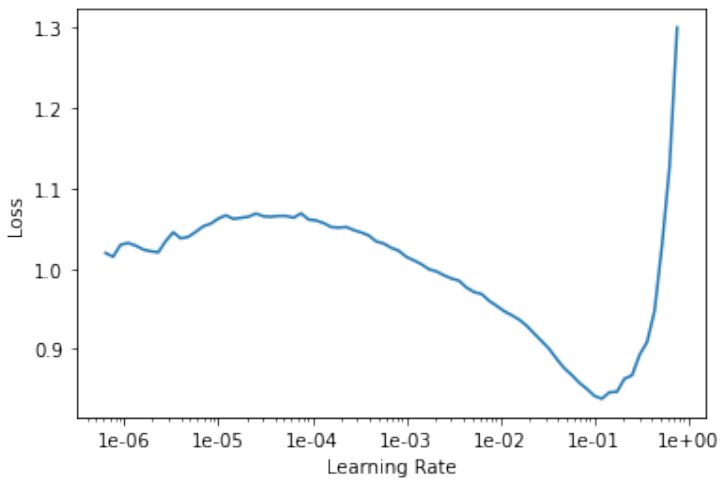

Figura 4.4: Tasa de aprendizaje obtenida de resnet50 al aplicar el databunch.

La tasa de aprendizaje (learning rate) permite optimizar el modelo, en la gráfica se puede observar el valor de la pérdida vs la tasa de aprendizaje, este valor de pérdida es la diferencia entre el resultado deseado y el resultado actual. Estos valores se obtienen al realizar validaciones con pequeños grupos distintos en el entrenamiento.

Se eligió una tasa de aprendizaje de **1e-02** en base a la Figura 4.4 porque los valores de la pérdida vienen disminuyendo desde tasas más pequeñas, pero al pasar el valor de 1e-01 vuelve a dispararse lo que significa que se producirá un sobreajuste. Con la tasa de aprendizaje en 1e-02 la pérdida está alrededor de 0.95 (ó 95 %), pero como se observa disminuye de una manera bastante rápida antes de llegar al sobreajuste.

- Una vez definida la tasa de aprendizaje se procede a realizar el entrenamiento y ajuste del modelo. Para ello, se estableció un número de 15 épocas para entrenar el modelo, es decir, la cantidad de veces que se realiza el entrenamiento con el conjunto de datos completo. Durante el ajuste se obtiene lo mostrado en las Figuras 4.5 y 4.6.

epoch	train_loss	valid_loss	accuracy_thresh	fbeta	time
0	0.905735	0.626958	0.487356	0.636721	03:00
1	0.756383	0.544985	0.564368	0.671277	03:00
2	0.557296	0.271979	0.859770	0.747794	02:59
3	0.414808	0.239727	0.885057	0.774011	03:01
4	0.320314	0.202409	0.901149	0.803977	02:59
5	0.270062	0.166493	0.927586	0.825787	03:01
6	0.243021	0.173843	0.918391	0.847266	02:59
7	0.218557	0.156888	0.921839	0.849356	02:59
8	0.195876	0.140276	0.942529	0.879775	03:01
9	0.183111	0.120645	0.943678	0.898700	02:59
10	0.159149	0.116830	0.944828	0.900499	02:59
11	0.146984	0.115294	0.955172	0.903460	02:59
12	0.128091	0.113878	0.952874	0.897306	03:03
13	0.118117	0.113886	0.951724	0.887960	03:01
14	0.107602	0.106790	0.955172	0.911587	03:00

Figura 4.5: Métricas obtenidas en el entrenamiento.

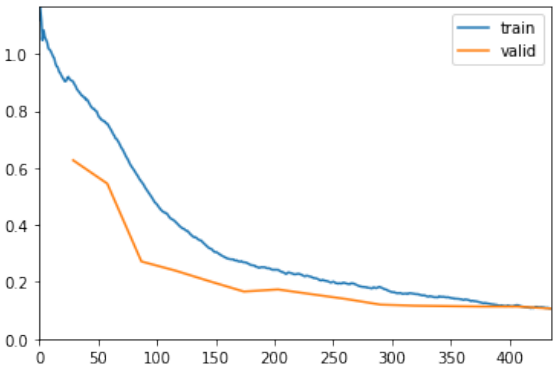

Figura 4.6: Comportamiento de la pérdida en el entrenamiento.

En la Figura 4.5 se observan las métricas establecidas.

- La exactitud comienza con un valor de 0.487 y conforme se avanza en el entrenimiento va aumentando rápidamente hasta llegar a un 0.955.
- La otra métrica es fbeta, como se ha dicho es un promedio entre la precisión y el recall. Tiene un comportamiento similar a la exactitud, comienza con un valor de 0.637 y aumenta a 0.912.

Mientras que en la Figura 4.6 se observa el comportamiento de la pérdida en el ajuste ya con el databunch de billetes. Comienza con una pérdida bastante grande de 0.905 y 0.627, en el entrenamiento y validación, respectivamente. En ambos la pérdida disminuye a un valor de 0.10.

Si los resultados obtenidos se comparan con la Figura 4.4, se observa un comportamiento similar desde la tasa de aprendizaje en 1e-02. Ya que en un inicio se tiene valores poco favorables como se ha presentado, pero conforme se ha avanza en las épocas del entrenamiento se llegan a obtener mejor resultados en todos los casos.

Además, se puede apreciar que el entrenamiento puede durar menos épocas, 10 para ser exacto. Las épocas siguientes consiguen resultados muy parecidos entre ellos.

En la Figura 4.7 se presenta las predicciones obtenidas después del primer entrenamiento, como se observa los resultados en la mayoría de los casos está correcto. El modelo falla en casos cuando se tiene dos billetes similares totalmente extendidos y en los casos que se ha tenido poca información como el caso de billetes de 200 €.

En la parte superior de cada imagen de los billetes de la Figura 4.7 se muestra lo qué es y su predicción, respectivamente.

Figura 4.7: Predicciones realizadas después del entrenamiento.

- El siguiente punto es realizar mejoras al modelo. Para esto se utilizan los dos pasos anteriores, se debe obtener una tasa de aprendizaje y luego realizar un ajuste.

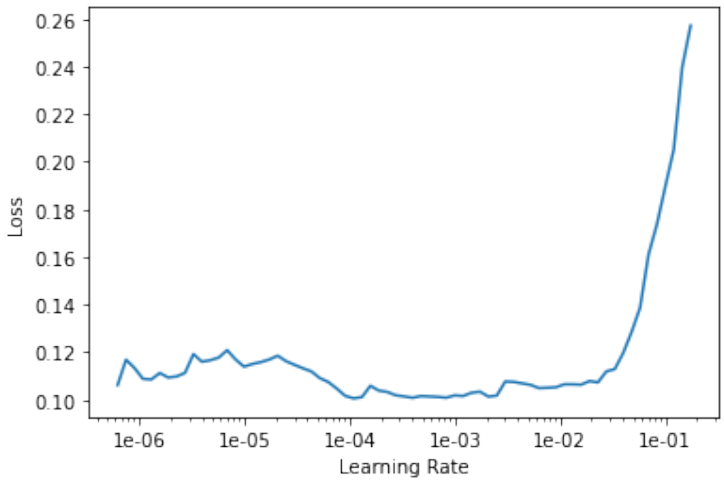

Figura 4.8: Tasa de aprendizaje obtenida después del primer entrenamiento.

Entonces primero se obtiene la tasa de aprendizaje de la Figura 4.8, donde el comportamiento de la curva de pérdida varía entre 0.10 y 0.12 comenzando con una tasa de aprendizaje de 1e-06 hasta llegar a 1e-02, después de esto el modelo se sobreajusta.

Por lo tanto, para el primer ajuste del modelo se comenzó con una tasa de aprendizaje en un rango entre 1e-04 y 1e-03, ya que en este rango se encuentran los valores más bajos de la pérdida.

Al definir un rango, el modelo se entrena de la siguiente manera. La tasa de aprendizaje igual a 1e-04, esta destinada para la capa superior, 1e-03 es la tasa aplicada en la capa inferior y las capas intermedias tomarán valores entre el rango dicho. Esto se conoce como aprendizaje discriminativo [3].

Los resultados del ajuste se observan en las Figuras 4.9 y 4.10, utilizando solo 10 épocas. Lo primero que determina es que los posteriores ajuste se pueden dar solamente con 5 épocas ya que después de la sexta época se consiguen valores similares. Estos resultados no han mejorado con respecto a los obtenidos del primer entrenamiento.

epoch	train_loss	valid_loss	accuracy_thresh	fbeta	time
0	0.096583	0.112590	0.954023	0.901370	03:01
1	0.109292	0.108006	0.955172	0.898990	03:01
2	0.109565	0.101623	0.958621	0.913271	03:00
3	0.100734	0.105291	0.957471	0.908859	03:00
4	0.096869	0.102798	0.959770	0.915186	03:02
5	0.098866	0.106875	0.958621	0.911529	03:00
6	0.097387	0.101245	0.957471	0.905027	03:02
7	0.093900	0.098878	0.957471	0.908684	03:01
8	0.089832	0.097905	0.958621	0.913387	02:59
9	0.092247	0.099411	0.956322	0.904737	03:01

Figura 4.9: Métricas obtenidas en el primer ajuste.

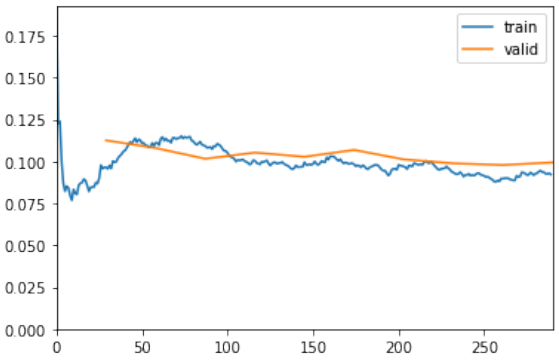

Figura 4.10: Comportamiento de la pérdida en el primer ajuste.

Al no tener cambios significativos, los parámetros fueron variados: la tasa de aprendizaje a un rango entre 1e-05 y 1e-03, y las épocas únicamente 5. Los resultados se presentan en las Figuras 4.11 y 4.12.

epoch	train_loss	valid_loss	accuracy_thresh	fbeta	time
0	0.072728	0.101056	0.950575	0.900035	04:01
1	0.118082	0.190194	0.940230	0.863927	04:00
2	0.128209	0.084871	0.970115	0.942355	03:59
3	0.105706	0.071313	0.970115	0.947899	04:00
4	0.098252	0.064445	0.972414	0.949379	04:00

Figura 4.11: Métricas obtenidas variando parámetros del primer ajuste.

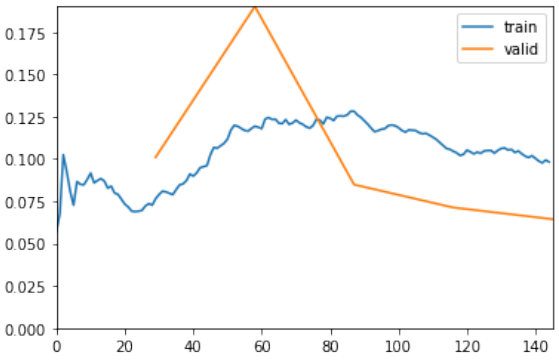

Figura 4.12: Comportamiento de la pérdida variando parámetros del primer ajuste.

[3] https://docs.fast.ai/basic_train.html#Discriminative-layer-training

Las mejoras se observan al comparar las Figuras 4.9 y 4.11. El valor de la pérdida en la primera está en 0.1, en la segunda ha bajado para el entrenamiento a 0.09 y para la validación 0.06, lo cual es una mejora alrededor del 60 %. Por otro lado, las métricas de éxito, en el ajuste se obtuvieron 0.972 y 0.949, para la exactitud y fbeta, respectivamente. Estos valores mejoraron frente a lo conseguido en el entrenamiento, 0.955 y 0.912, para las métricas mencionadas.

En la Figura 4.13 se presenta las predicciones realizadas. Al comparar con la Figura 4.7, el modelo después del primer entrenamiento para el billete de 200 euros predijo los valores de 200 y 50, acertando únicamente en que son euros. Después del ajuste el modelo a la misma imagen la clasifica como 200 euros.

Además, en ambas figuras la primera imagen (esquina superior izquierda) en realidad es un billete de 20 dólares la cual por error ha sido marcada como 5 dólares desde la creación de la fuente de datos, a pesar de ello el modelo la ha clasificado correctamente, sin importar que esté definida con una etiqueta errónea.

La única falla en la Figura 4.13 se encuentra en la imagen ubicada en la segunda fila y tercera columna, el billete de 10 dólares no ha sido reconocido correctamente. El modelo únicamente ha podido reconocer su denominación y no su valor.

Figura 4.13: Predicciones realizadas después del ajuste.

Por otro lado, al analizar la Figura 4.10, las curvas de pérdida tanto en entrenamiento como validación tienden a disminuir. Por este motivo, se analiza la posibilidad de un segundo ajuste para determinar si el modelo puede mejorar aún más. Con ayuda de la Figura 4.14 se observa la curva de la tasa de aprendizaje que tiene el modelo hasta el momento, en ella se aprecia que conforme disminuye la tasa de aprendizaje, el valor de la pérdida solamente tiende a subir lo que significa que el modelo se sobreajustará si se entrena nuevamente. Por este motivo, se ha establecido ocupar el modelo solamente con el primer ajuste ya que se han conseguido las métricas establecidas con valores superiores al 95 %.

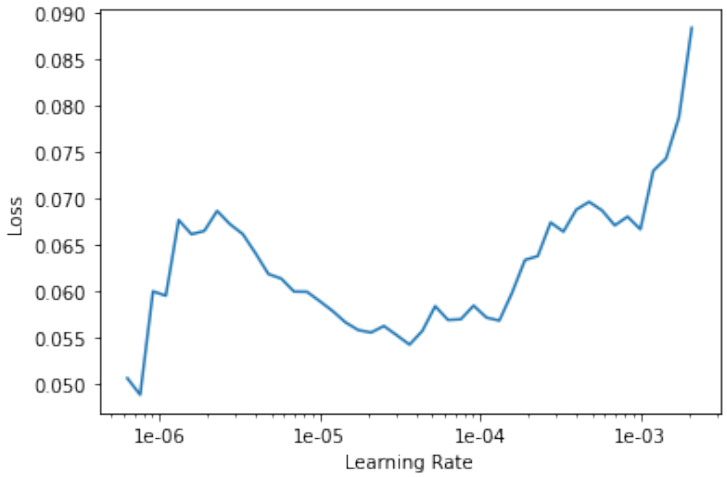

Figura 4.14: Tasa de aprendizaje obtenida después del primer ajuste.

4.3.3. Llevando el modelo a Amazon S3

De acuerdo a la literatura, la aplicación desarrollada con SAM espera que un modelo PyTorch en formato TorchScript, esté almacenado en S3 junto con un archivo de texto de clases (etiquetas), el cual contiene los nombres de todas las posibilidades de salida, para ser utilizado posteriormente en la función Lambda.

Una vez con el modelo en el formato adecuado para trabajar en el entorno descrito de Lambda, se procede a almacenarlo en el bucket anteriormente creado en S3. Para esto se utilizan las credenciales creadas, la cuales permiten conectarse a la cuenta de la persona quien desarrolla el proyecto.

Cabe mencionar que todo lo mostrado hasta el momento, es implementado en Google Collab, una herramienta en línea que permite la ejecución de código Python. En el código mostrado en el Apéndice B.1, la sección final está dedicada para subir el modelo al bucket. Lo que se ha hecho es definir las credenciales para conectarse con S3, algo muy importante ya que no basta con solo definir el nombre del bucket. No es aconsejable ejecutarlo de la manera que se muestra por cuestiones de seguridad, pero por temas didácticos se lo ha realizado. Con el resto de código se realiza la conexión y posterior subida del modelo al bucket **tfm-bucket-fact** que fue creado y la función Lambda pueda hacer uso de él.

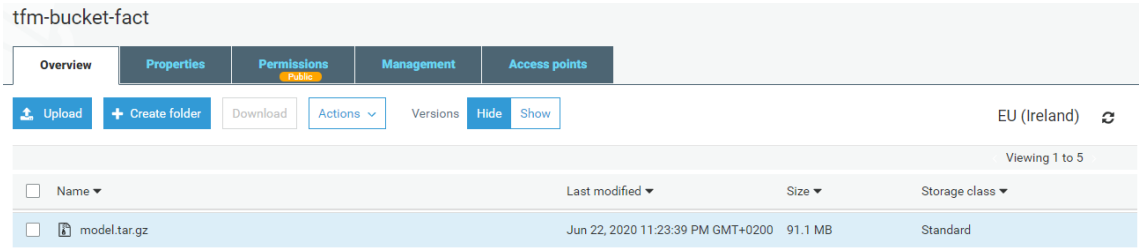

Figura 4.15: Modelo subido exitosamente a S3.

4.3.4. Predicciones del modelo, pruebas y resultados

Una vez obtenido el modelo, se procedió a realizar varias predicciones con un nuevo conjunto de 25 fotografías de billetes, tanto euros como dólares, en dos escenarios de

pruebas.

- El primer escenario consistió en tener varias fotografías, para ser exactos 13 del nuevo conjunto, donde los billetes están completamente estirados, al aplicar el modelo de clasificación se obtuvieron los resultados mostrados en la Tabla 4.1.

Imagen de Entrada		**Predicción**	**Tiempo (s)**	**Descripción del resultado**
Tipo	**Valor**			
euro	20	[20, euro]	0.20	Correcto
euro	50	[50, euro]	0.20	Correcto
euro	50	[euro]	0.19	Tipo de billete correcto
euro	50	[euro]	0.20	Tipo de billete correcto
euro	10	[euro]	0.20	Tipo de billete correcto
euro	500	[500, euro]	0.20	Correcto
euro	5	[5, euro]	0.20	Correcto
usd	20	[20, usd]	0.20	Correcto
usd	20	[20, usd]	0.22	Correcto
usd	20	[20, usd]	0.20	Correcto
usd	5	[5, usd]	0.20	Correcto
usd	5	[5, usd]	0.20	Correcto
usd	10	[10, usd]	0.20	Correcto
usd	10	[10, usd]	0.21	Correcto

Tabla 4.1: Resultados obtenidos en el primer escenario.

Como se observa la mayoría de las predicciones, específicamente un 76.9 % (10 de un total de 13) han sido correctamente identificadas, tanto en el tipo como el valor del billete. El 23.1 % (3 imágenes) restante solo ha clasificado correctamente el tipo.

En estos casos, el modelo permite conocer alternativas del resultado de clasificación. Para estas 3 imágenes las alternativas de clasificación son presentadas en la Tabla 4.2, donde el campo "predictions" tiene las probabilidades de cada clase y en el campo "others" se indica en porcentajes las clases con mayor probabilidad.

Imagen de Entrada		**Predicción**	**Alternativas**
euro	50	[euro]	"predictions": ["0.00", "0.02", "0.03", "0.00", "0.30", "0.00", "1.60", "0.01", "98.01", "0.03"], "others": {"50": "1.60 %", "200": "0.30 %", "100": "0.03 %", üsd": "0.03 %", "10": "0.02 %", "500": "0.01 %"}
euro	50	[euro]	"predictions": ["0.00", "0.00", "0.00", "0.00", "0.00", "0.00", "0.00", "0.00", "100.00", "0.00"], "others": {}
euro	10	[euro]	"predictions": ["0.00", "0.00", "0.00", "0.00", "0.01", "0.00", "0.00", "0.02", "99.98", "0.00"], "others": {"500": "0.02 %", "200": "0.01 %"}

Tabla 4.2: Alternativas de clasificación - Resultado del primer escenario.

Por ejemplo, en el primer caso la alternativa más alta indica que es un billete de 50, lo cual es correcto, pero en el segundo caso el modelo no ha podido identificar nada más que solo el tipo. Y en el tercer caso las alternativas mostradas no son correctas. Estos tres casos podrán darse en la implementación de la aplicación al no reconocer correctamente la imagen en este tipo de escenario.

- En el segundo escenario, las fotografías de los billetes se tomarán con un doblez, es decir, solo aparece la mitad del billete. Los resultados de esta prueba son mostrados en la Tabla 4.3 para lo cual se utilizaron 12 fotografías del nuevo conjunto.

Imagen de Entrada		**Predicción**	**Tiempo (s)**	**Descripción del resultado**
Tipo	**Valor**			
euro	20	[20, euro]	0.20	Correcto
euro	20	[20, euro]	0.20	Correcto
euro	20	[20, euro]	0.20	Correcto
euro	50	[euro]	0.20	Tipo de billete correcto
euro	10	[10, euro]	0.20	Correcto
euro	5	[500, euro]	0.20	Tipo de billete correcto Valor del billete incorrecto
usd	20	[20, usd]	0.20	Correcto
usd	20	[20, usd]	0.22	Correcto
usd	20	[20, usd]	0.20	Correcto
usd	5	[5, usd]	0.20	Correcto
usd	5	[5, usd]	0.20	Correcto
usd	10	[10, 100, usd]	0.20	Tipo de billete correcto Valor del billete incorrecto
usd	10	[10, usd]	0.21	Correcto

Tabla 4.3: Resultados obtenidos en el segundo escenario.

En este escenario los resultados son parecidos al primero, se tiene un porcentaje del 83.3 % de aciertos (10 de un total de 12), mientras que el 16 % restante no se ha podido identificar el valor, solo el tipo de billete.

Imagen de Entrada		**Predicción**	**Descripción del resultado**
euro	50	[euro]	“predictions”: ["0.00", "0.00", "0.00", "0.00", "0.01", "0.00", "0.00", "0.02", "99.98", "0.00"], “others”: {"500": "0.02 %", "200": "0.01 %"}
euro	10	[euro]	“predictions”: ["0.00", "0.06", "0.13", "0.00", "0.00", "0.00", "0.00", "0.00", "0.00", "99.81"], “others”: {}

Tabla 4.4: Alternativas de clasificación - Resultado del segundo escenario.

Las alternativas a estos dos casos sin identificar son presentadas en la Tabla 4.4, en la que se aprecia que en ambos casos las alternativas no son las correctas. En el primer caso, las probabilidades para identificar un valor son bajas y en clases equivocadas, mientras que el segundo caso como identifica dos valores, ya no presenta otras alternativas.

Para mejorar los resultados se debería realizar un entrenamiento con un nuevo conjunto de imágenes lo cual ayudará al modelo a reconocer los billetes en caso de que estén doblados. Y la opción más rápida para que el asistente reconozca el billete sería estirarlo completamente y hacer una nueva fotografía.

4.4. Desarrollo de función en Lambda como Serverless

4.4.1. Estructura del proyecto

La estructura que maneja AWS SAM para cualquier proyecto contiene dos ficheros importantes. Los cuales son descritos a continuación ya que cumplen funciones específicas para el desarrollo. Los nombres de cada archivo pueden ser cambiados por quién desarrolla la aplicación, al cambiar se deberá tener en cuenta en posteriores llamados.

- **app.py**: Contiene la lógica del controlador Lambda en lenguaje Python.
- **template.yaml**: Es la plantilla que AWS SAM utiliza para crear y desplegar los recursos de AWS de la aplicación.

4.4.2. Configuración del template

Se ha configurado la función Lambda para extraer código y contenido adicional en formas de capas Lambda. Una capa es un archivo ZIP que contiene bibliotecas, un tiempo de ejecución personalizado u otras dependencias. Con las capas, se pueden utilizar bibliotecas en función de la necesidad sin incluirlas en el paquete de implementación.

En este proyecto, se utiliza una capa Lambda acceso público que contiene las bibliotecas PyTorch necesarias para ejecutar la aplicación. Estas capas se implementan en las siguientes regiones: us-west-2, us-east-1, us-east-2, eu-west-1, ap-sureste-1, ap-sureste-2, ap-noreste-1, eu-central-1. La región predeterminada es us-east-1. Hay 2 versiones de la capa Lambda PyTorch con diferentes versiones de PyTorch que se muestran en la Tabla 4.5.

Capa ARN	**Versión PyTorch**
arn: aws:lambda: **AWS_REGION**: 934676248949: capa: pytorchv1-py36: 1	PyTorch 1.0.1
arn: aws:lambda: **AWS_REGION**: 934676248949: capa: pytorchv1-py36: 2	PyTorch 1.1.0

Tabla 4.5: Versiones de PyTorch.

Aquí se debe tener en cuenta un concepto interesante de AWS, que son las *regiones* y se define como una ubicación física en todo el mundo donde se agrupan los centros de datos. A cada grupo de centros de datos lógicos se lo conoce como zona de disponibilidad (AZ). Cada región de AWS consta de varias AZ aisladas y separadas físicamente dentro de un área geográfica. La selección de una región cercana es importante porque permite tener mayor rapidez en la conexión de los recursos, además los recursos creados en determinada región pueden ser compartidos de manera más sencilla. En algunos casos, el tener recursos en distintas regiones de AWS no permite interactuar entre ellos debido a restricciones.

Por ejemplo en la sección 4.2.2, cuando es creado el bucket se define la región, EU

(Ireland). Para AWS esta región se puede encontrar como eu-west-1, la cual fue seleccionada por ser la más cercana a España y estar dentro las regiones para ejecutar la capa pública de PyTorch.

Entonces en el fichero **template.yaml** se debe implementar los parámetros, variables, recursos, salidas y capas que conforman la anatomía[4] del template y son descritos a continuación. La implementación final se observa en el apéndice B.2.

- **Parámetros:** Para lograr una conexión con S3, se debe configurar los parámetros **BucketName** y **ObjectKey** ya que en ellos se definen los nombres del bucket (tfm-bucket-fact) y del objeto (model.tar.gz) a ser utilizados por Lambda.

  Para conectar con la biblioteca de PyTorch, en el parámetro **LambdaLayerArn** se define la segunda capa mostrada en la Tabla 4.5 con la región eu-west-1, permitiendo facilidad y rapidez al momento de conectar recursos en una misma región.

- **Variables globales:** Con las variables globales se pueden definir propiedades comunes a todas las funciones serverless. Por ejemplo, para este proyecto se configura un tiempo de desconexión (timeouts) el cual queda definido con 300 milisegundos.

- **Recursos:** En esta sección del template se indica la función principal a ejecutarse, el lenguaje de programación junto con la versión requerida, el tamaño de memoria a utilizarse y variables de entorno como por ejemplo el modelo de clasificación.

  Algunos parámetros son tomados con referencias indirectas de los campos anteriormente definidos. También se indica el recurso utilizado de la API Gateway para responder ante eventos. Se debe tener en cuenta la estructura del proyecto, en este caso la función principal esta en otro directorio y con ayuda de *CodeUri* permite indicar esta ruta.

- **Outputs:** Con esta sección se define la interfaz de comunicación que tendrá la función Lambda. A través del recurso API Gateway se crea una API la cual se soporta de los recursos previamente definidos para ser utilizada como un microservicio.

4.4.3. Descripción de la función principal *lambda_handler*

Para tener una visión global de lo programado se presenta la Figura 4.16.

La función en Lambda realiza inferencias de imágenes (peticiones realizadas) con ayuda de un modelo de visión por computadora. Cuando se invoca la función, primero se descarga el modelo PyTorch de S3 y luego pasa a ser cargado en la memoria de la función para ser utilizada.

Desde algún lugar de internet se envía la petición para conocer la cantidad y denominación del billete. Esta función recibe la imagen como entrada por medio de un objeto JSON el cual contiene la URL de ésta. La función descarga la imagen, convierte sus píxeles en un objeto tensor de PyTorch y la pasa a través del modelo PyTorch.

Lo que se devuelve es el nombre de la clase (cantidad y denominación) con el puntaje de salida más alto del modelo. En ciertos casos las imágenes tendrán características similares a pesar de ser de distintos valores, en estos casos se devuelve un nivel de confianza o probabilidades de estar en otras clases.

La programación se centra en la función principal *lambda_handler* ya que como se menciona en la literatura expuesta anteriormente, Lambda utiliza el código desarrollado

[4] `https://docs.aws.amazon.com/serverless-application-model/latest/developerguide/sam-specification-template-anatomy.html`

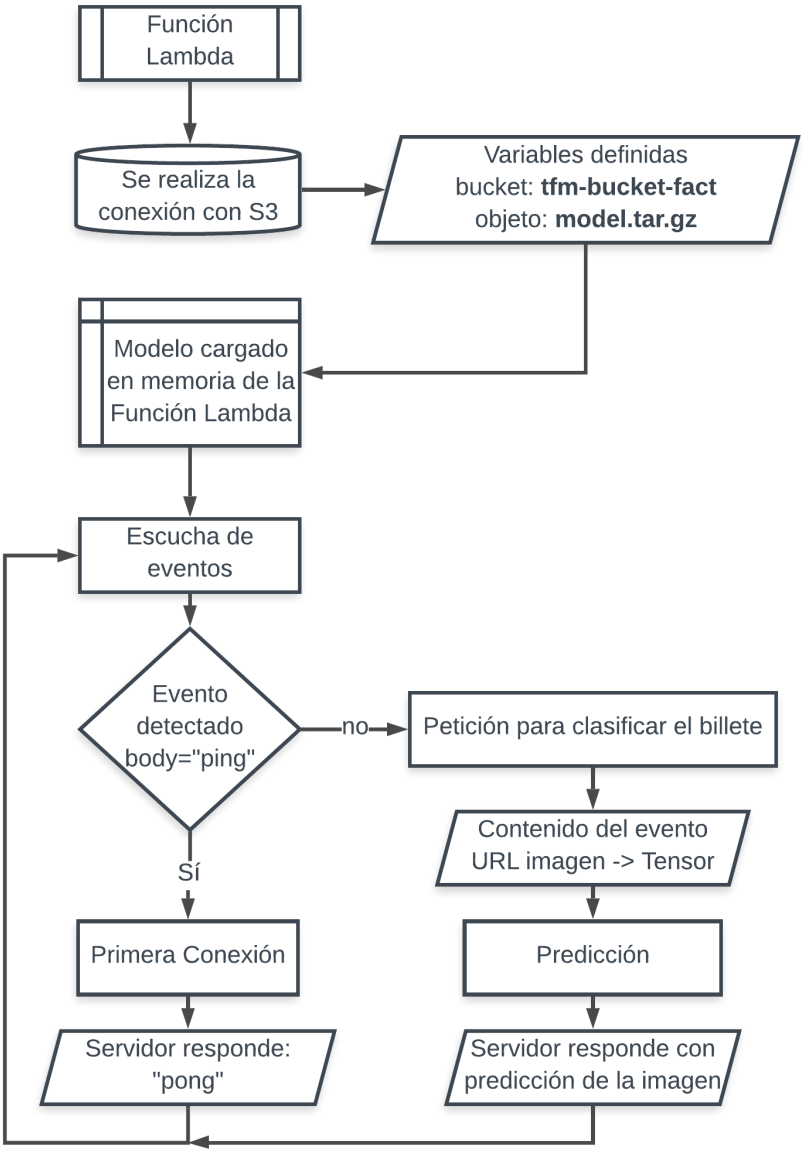

Figura 4.16: Diagrama de flujo de la función Lambda.

en cualquier lenguaje y el único cambio a tener en cuenta al pasarlo a este servicio es en la función principal ya que será diferente. Lambda utiliza esta función como controlador y a través de ella escucha eventos que serán los disparadores para que la función se active y realice las distintas tareas. Este desarrollo fue en base a la documentación presentada por AWS Lambda[5] enfocada al uso de Python y su controlador.

Se puede observar la implementación de la programación dentro del Apéndice B.2.

4.4.4. Despliegue de la función Lambda con SAM

En el Apéndice A.2 se presenta como se debe establecer el entorno para utilizar AWS SAM ya que una vez configurado el template y la función principal, se deberá empaquetar la función y posteriormente desplegar el microservicio en AWS.

[5] `https://docs.aws.amazon.com/es_es/lambda/latest/dg/python-handler.html`

- Primero se empaqueta la función Lambda y se almacena en el bucket creado con el comando[6] ejecutado y mostrado en la Figura 4.17.

Figura 4.17: Empaquetado y almacenamiento de la función Lambda en S3.

- Luego, se procede a desplegar la aplicación con el comando mostrado en la Figura 4.18. Este comando creará una pila con ayuda de Cloudformation (otro servicio de AWS) y desplegará los recursos descritos en el template, lo que se observará a lo largo de la ejecución. En la parte final se observará la interfaz de salida creada para el microservicio.

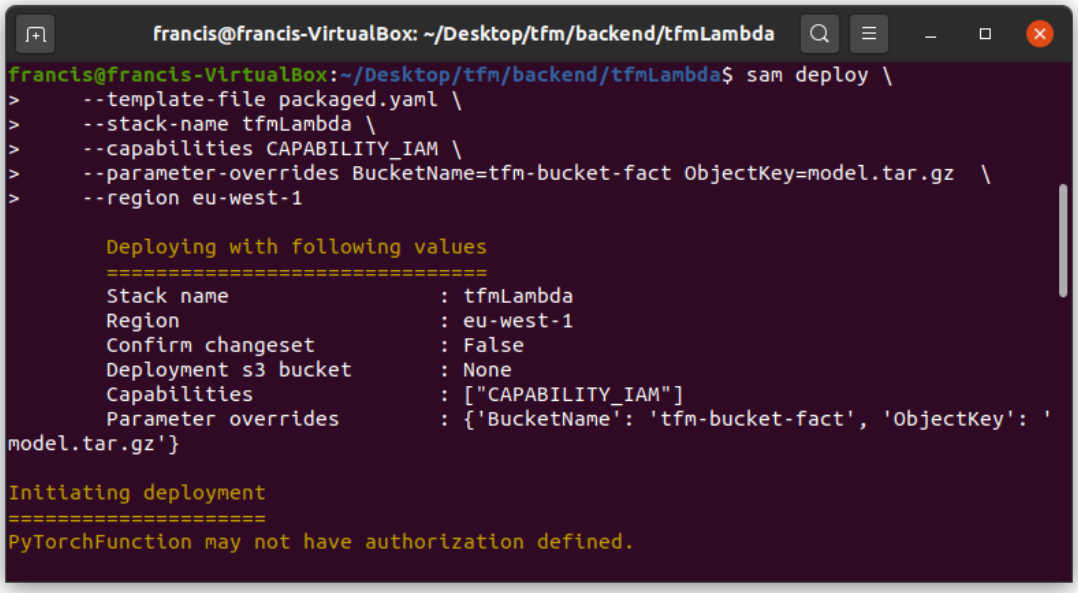

Figura 4.18: Despliegue de la función Lambda empaquetada anteriormente.

- Al completarse la implementación, se ejecuta el comando[7] mostrado en la Figura 4.19 el cual recupera la URL del endpoint de la API Gateway. Este endpoint será utilizado

[6] https://docs.aws.amazon.com/serverless-application-model/latest/developerguide/sam-cli-command-reference-sam-package.html

[7] https://docs.aws.amazon.com/cli/latest/reference/cloudformation/describe-stacks.html

dentro de la aplicación como destino para enviar las peticiones con las imágenes fotografiadas.

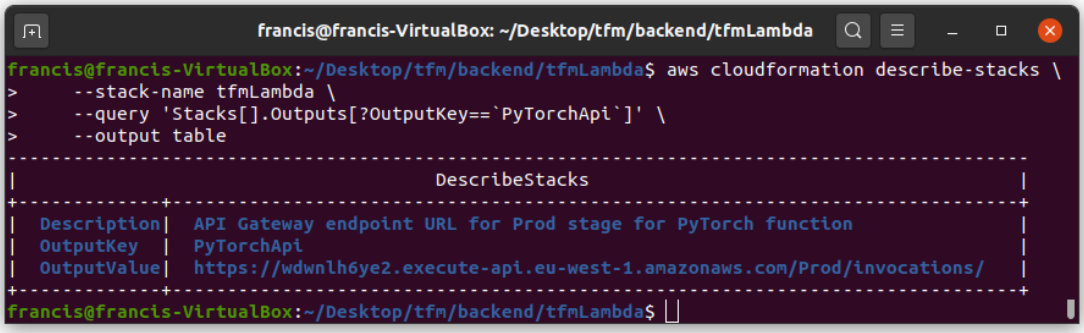

Figura 4.19: Endpoint creado en el despliegue de Lambda

Una vez finalizado el despliegue, se puede verificar en la consola de administración (entorno gráfico) de AWS que los dinstintos recursos han sido creados como se observa en la Figura 4.20.

Logical ID	Physical ID	Type
PyTorchFunction	tfmLambda-PyTorchFunction-2CZGAZW4ZY8D	Lambda Function
PyTorchFunctionPyTorchPermissionProd	tfmLambda-PyTorchFunctionPyTorchPermissionProd-1DOLDIH05F673	Lambda Permission
PyTorchFunctionRole	tfmLambda-PyTorchFunctionRole-JYWLIVIYGC8J	IAM Role
ServerlessRestApi	wdwnlh6ye2	ApiGateway RestApi
ServerlessRestApiDeployment8814ce52ef	fow04b	ApiGateway Deployment
ServerlessRestApiProdStage	Prod API endpoint	ApiGateway Stage

Figura 4.20: Función Lambda implementada vista desde la consola de AWS.

En la Figura 4.21 se observa la ejecución de las primeras peticiones realizadas a este microservicio. AWS permite obtener varias métricas ya que es una característica de la nube tener monitoreado los servicios utilizados, pero se debe tener en cuenta que algunas métricas incurrirán en un pago. Las métricas presentadas están de manera gratuita.

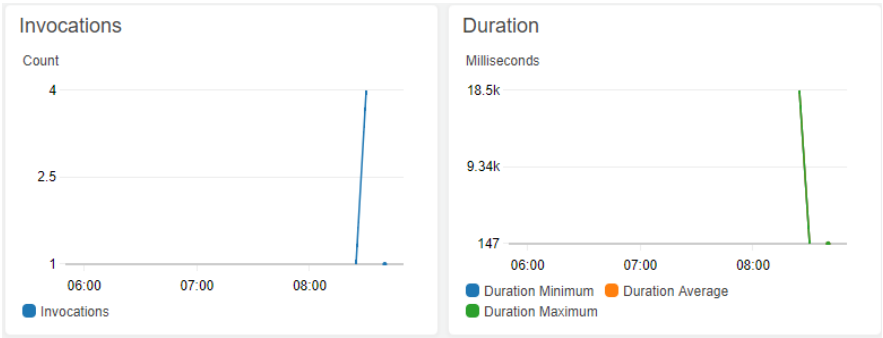

Figura 4.21: Peticiones realizadas a la función Lambda desde el cliente.

4.5. Desarrollo del cliente

Esta sección esta dedicada para describir la implementación de la aplicación, para ello primero se detallan los pasos para crear el proyecto junto con los complementos que se requiere, como por ejemplo la cámara. Luego se presenta la aplicación en su estado inicial cumpliendo los criterios de diseño y accesibilidad vistos en la sección 3.8, también se presenta la aplicación ya en ejecución. Por último, son presentadas algunas pruebas realizadas durante la implementación para observar la interacción que tiene la aplicación con los diferentes microservicios en la nube, para lo cual se hace uso del despliegue de la aplicación en un entorno web. El código desarrollado para el cliente se puede observar en Apéndice B.3.

4.5.1. Creación del proyecto

Para el desarrollo de la aplicación, se debe recordar lo realizado en el Apéndice A.1.2, al ejecutar el contenedor con Apache Cordova, se establece un ruta del directorio local para compartir una carpeta, teniendo todo el desarrollo en local y no se pierda al detener el contenedor.

Se ejecuta el contenedor con Cordova, para crear y configurar el proyecto ejecutando los siguientes comandos.

```
cordova create tfmApp com.example.tfmApp tfmApp
cd tfmApp
cordova platform add android
cordova platform add browser
cordova plugin add cordova-plugin-camera
cordova plugin add cordova-plugin-whitelist
cordova plugin add cordova-plugin-file
```

Después de esto, en la dirección */home/francis/Desktop/tfm/frontend*, indicada al arrancar el contenedor se puede observar lo creado y añadido hasta el momento.

Cuando se termine de realizar las distintas tareas en el contenedor, se recomienda salir y detener el contenedor con el comando `docker stop idContenedor`. Mientras se realicen cambios durante la preparación del proyecto, se sugiere mantenerse en el modo iterativo del contenedor ya que se debe construir la aplicación.

4.5.2. Descripción de los ficheros utilizados para la aplicación

El proyecto creado se encuentra en la ruta mencionada, dentro de la carpeta **tfmApp** están los ficheros que permiten el desarrollo del cliente con los lenguajes de programación **HMTL5+CSS3+Javascript**. Al crear el proyecto, Cordova por defecto crea una plantilla para comenzar a trabajar. En la Figura 3.12 se observa como la combinación de estos lenguajes y a través de un entorno web permite desarrollar una aplicación para un entorno móvil.

- www/index.html es la interfaz hacia el usuario la cual se codifica mediante HTML5 permitiendo así distribuir la aplicación mediante componentes cada sección requerida.

  En la Figura 4.22 se observa la implementación de las directrices de diseño presentadas en la sección 3.8, esta figura muestra la aplicación en su estado inicial. Como se explicó en el diseño cada una de las tres secciones cumple una función en específica;

la primera es para desplegar la fotografía tomada, la segunda para mostrar la información de la clasificación y cambio de divisa, y la tercera accede a la cámara para capturar fotografías a los billetes.

Como se puede observar, la aplicación en un estado inicial solo muestra 2 secciones ya que no tiene información para desplegar. Estas dos secciones tienen descripciones cortas las cuales el usuario objetivo de este proyecto podrá identificar de manera clara y precisa al hacer uso de TalkBack, el asistente para accesibilidad. TalkBack indicará que la primera es para visualización y la siguiente es para capturar fotografías si se da un doble pulso sobre la pantalla.

Por otro lado, en la Figura 4.23 se observa la aplicación funcionando, en ella se tienen las 3 secciones inicialmente descritas. El fin de la aplicación es tomar una fotografía, identificarla y mostrar el cambio que tiene en ese momento. También, en caso de no tener una certeza de la imagen identificada, se presenta las alternativas de los billetes que puede ser junto con sus respectivas probabilidades.

El asistente de accesibilidad TalkBack ayuda al usuario a saber su resultado ya que por medio de voz y al seleccionar la segunda sección describirá todo lo que hay en ella.

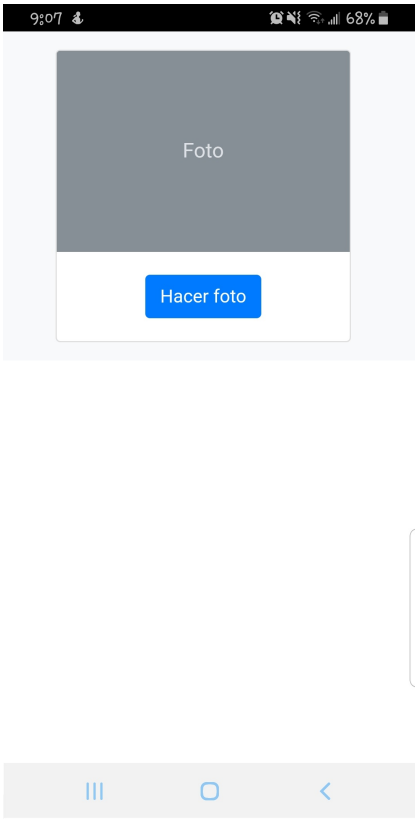

Figura 4.22: Aplicación en su estado inicial.

Figura 4.23: Aplicación en ejecución.

- www/css/index.css, permite modificar características con código CSS3 a los objetos mostrados en la interfaz. Se relaciona con index.html mediante etiquetas.

  La implementación de este código igualmente se puede observar en las Figuras 4.22 y 4.23. Cada sección tiene un color, tamaño y formato distinto, esto se logra con ayuda de index.css.

- www/js/index.js, este fichero tiene el código Javascript y ejecuta lo dinámico de la aplicación. Permite la interacción con la cámara para tomar esta información y enviarla al microservicio respectivo. Las funciones son asociadas con elementos del fichero index.html, en este caso específico al aplastar el botón "Hacer foto" se hace el llamado para abrir la cámara y posteriormente ejecutar los microservicios.

  Para acceder a la cámara se utilizó las estructuras facilitadas en la documentación[8] de Cordova, las cuales puede ser implementadas al haber configurado el complemento de la cámara en la parte inicial del proyecto.

  Para ejecutar los microservicios, se hace uso de peticiones AJAX que son un conjunto de técnicas de desarrollo web para el intercambio de datos de forma asíncrona que permite procesar cualquier solicitud a servidores en segundo plano. La ejecución de las peticiones AJAX tiene el siguiente orden:

  1. El cliente crea una llamada de JavaScript que luego activará XMLHttpRequest.
  2. En segundo plano, el cliente web crea una solicitud HTTP al servidor.
  3. El servidor recibe, recupera y envía los datos al cliente.
  4. El cliente recibe los datos solicitados que aparecerán directamente en la aplicación. No se necesita recargar.

  Cabe recalcar que el cliente realmente es el entorno web o navegador web desplegado como aplicación móvil.

  Esto se realiza para el microservicio desplegado en Lambda y el servicio de cambio de divisas, las peticiones son POST y GET, respectivamente. La petición POST se utiliza porque se requiere que la información enviada al servidor sea procesada para identificar el billete, mientras que la petición GET solo solicita la información al servidor del cambio de divisa actual entre el euro y el dólar.

4.5.3. Despliegue de la aplicación

Una vez acabada la programación, se debe construir la aplicación para que se genere el fichero que permite instalar la aplicación en el teléfono móvil. La construcción es muy sencilla, dentro del contenedor se debe ejecutar el siguiente comando

```
cordova build
```

El comando lo que hace es construir la aplicación para todas las plataformas configuradas inicialmente, aquí se observa la ventaja de desarrollo con Apache Cordova ya que con un mismo código se logra desplegar la aplicación para una plataforma web y para una plataforma móvil como es Android.

El fichero que se requiere para instalar la aplicación en el móvil se encuentra en la siguiente ruta dentro del proyecto */tfmApp/platforms/android/app/build/outputs/apk/debug/* con el nombre **app-debug.apk**.

La construcción para la plataforma web es utilizada para realizar pruebas ya que al ejecutar la aplicación en un navegador se puede hacer uso de la consola lo que permite observar la interacción de los microservicios. Dentro del contenedor se debe ejecutar el siguiente comando para observar la aplicación web en el navegador.

```
cordova run browser
```

[8] `https://cordova.apache.org/docs/es/3.1.0/cordova/camera/camera.getPicture.html`

Una vez ejecutado el comando, se indicará una dirección a la cual acceder, por lo general suele ser localhost:8000/index.html. En este caso, al utilizar contenedores la dirección que sustituye a localhost es 172.17.0.2, esto dependerá la asignación de IP que se haga al contenedor desde la máquina.

Al abrir un navegador web e ir a la dirección mencionada se observa la aplicación como en la Figura 4.24 en la cual se muestra el estado inicial y con ayuda de la consola se observa la primera ejecución del microservicio con Lambda para el ejecución en frío descrito en la sección 3.9. Las Figuras 4.22 y 4.24 muestran la misma aplicación, pero en diferentes plataformas.

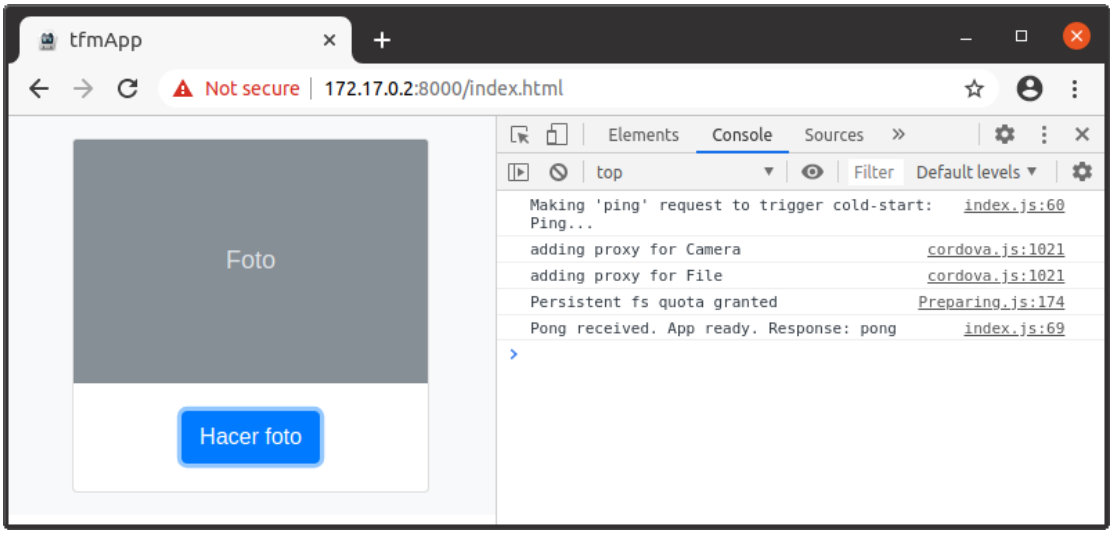

Figura 4.24: Aplicación en su estado inicial en un entorno web.

En la Figura 4.25 se presenta la ejecución de la aplicación, se envía una fotografía al servicio de Lambda para que identifique el billete, recibir la respuesta. Luego solicitar el valor de cambio de divisa del momento y finalmente presentar el resultado obtenido con el cambio de divisa correspondiente. En esta figura se observa la respuesta de Lambda, se obtiene un resumen de lo identificado ya que pueden darse casos que no se identifique correctamente y se desplieguen varias alternativas. También se observa la petición del cambio de divisa, a fecha 14 de Julio, se tiene un cambio de 1.1448 dólares por euro.

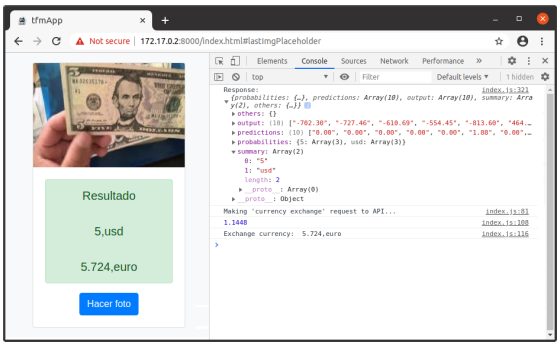

Figura 4.25: Ejecución de la aplicación en un navegador web - Prueba con 5$.

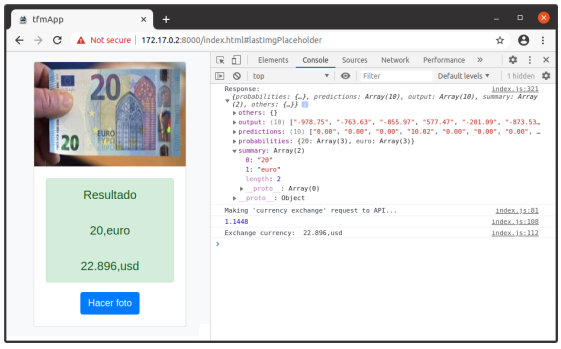

Figura 4.26: Ejecución de la aplicación en un navegador web - Prueba con 20€.

En la Figura 4.26 es presentado el mismo ejercicio anterior pero en este caso es con un billete de 20 euros, el caso anterior fue con un billete de 5 dólares. Como se puede

observar tiene la misma interacción con los microservicos, lo único que cambia es el cambio de divisa. El microservicio para esta función entrega el cambio de euro a dolar, con ayuda de la etiqueta identificada se procede hacer el cálculo respectivo y finalmente se presenta.

Como se presentó en la sección 4.3.4, el modelo a pesar de que en la mayoría de los casos identifica correctamente las imágenes, suele en algunos otros no hacerlo. En ese caso, la aplicación la información como en la Figura 4.27, donde no se puede identificar correctamente el valor, pero sí su denominación.

Esta prueba ha sido complicada ya que tiene 3 billetes del mismo tipo y valor, pero el modelo solo ha logrado identificar el tipo. Además, se muestra como posible alternativa el valor de 5 que pudiera ser la imagen, pero al tener la probabilidad muy baja no es asignado.

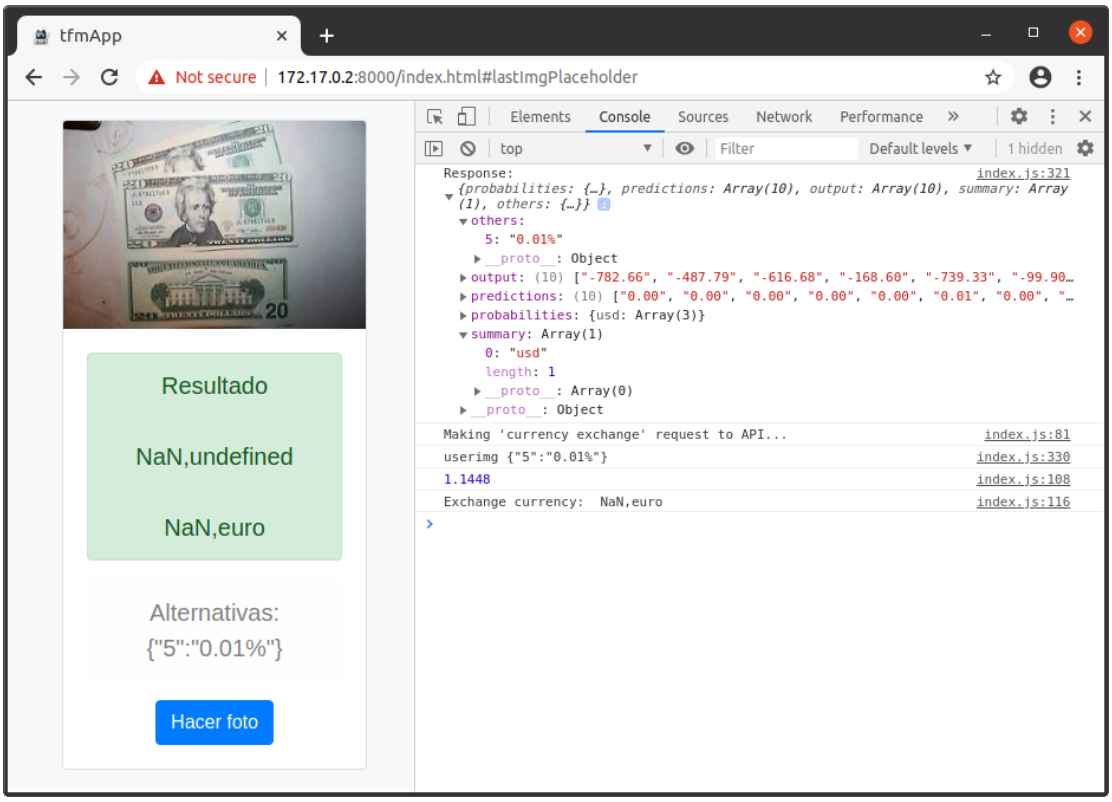

Figura 4.27: Ejecución de la aplicación en un navegador web - Prueba con varios billetes.

Capítulo 5

Conclusiones y Trabajo Futuro

"Just because something doesn't do what you planned it to do doesn't mean it's useless."
— Thomas Edison

Este capítulo se enfoca en presentar las conclusiones y trabajos futuros en relación al presente trabajo, haciendo una retrospectiva de lo realizado y teniendo en cuenta que todo esta bajo el paradigma de IoT. Con respecto a los objetivos planteados al inicio, se han alcanzado todos.

5.1. Conclusiones

El principal objetivo de este proyecto era desarrollar un asistente accesible que permita la identificación de billetes y el cambio de divisa en tiempo real con conceptos y tecnlogías de IoT. Este asistente fue desplegado para teléfonos móviles con sistema operativo Android haciendo uso de TalkBack como herramienta de accesibilidad para su fácil uso e interacción con el usuario. También se integraron recursos y servicios en la nube dentro de este asistente.

Se debe mencionar que el asistente accesible desarrollado es una prueba de conceptos, ya que permitió integrar varias tecnologías y herramientas del IoT como sensores, inteligencia artificial, microservicios, aplicaciones multiplataforma, servicios en la nube, etc.

Otro de los objetivos del proyecto era implementar un modelo de inteligencia artificial que permita la identificación de billetes. Para esto se desarrollo un modelo que tiene como base a resnet50, una red neuronal pre entrenada con 14 millones de imágenes y facilitada en la biblioteca de fast.ai. Para ajustar la red al problema planteado se formó un conjunto de entrenamiento con 440 imágenes de billetes con distintos valores entre euros y dólares, logrando así una exactitud y precisión alta, alrededor del 95 %. Además, el modelo fue puesto a prueba con un conjunto de prueba distinto al del ajuste inicial, obteniendo un acierto en la mayoría de sus predicciones como se mostró en las Tablas 4.1 y 4.3. Hubieron casos en los que no se pudo determinar correctamente el billete, para ello se mostraban alternativas de la predicción como las mostradas en las Tablas 4.2 y 4.4.

También, a lo largo de este documento se presentó el diseño y la implementación de la infraestructura para el desarrollo del asistente accesible, la cual emplea conceptos de IoT presentados en el estado del arte y un dispositivo IoT muy común en el día a día de las personas que es el teléfono móvil para darle un valor añadido al explotar los recursos que viene con él. En base a la infraestructura implementada se concluye lo siguiente:

- **La ejecución de las tareas han sido repartidas eficientemente**

  El teléfono móvil utilizado como dispositivo IoT ejecuta el asistente o aplicación desarrollada la cual interactúa con la cámara para fotografiar los distintos billetes a ser procesados. Además, presenta la información obtenida de la identificación y cambio de divisa, con ayuda del asistente de accesibilidad TalkBack que reproduce por voz el texto desplegado. Todo esto se resume como un cliente.

  Por otro lado, esta la tarea de identificación de billetes la cual es procesada en la nube con ayuda de AWS Lambda que ejecuta el modelo de clasificación almacenado en S3, realiza las predicciones y envía la información resultante. También está la tarea de conocer el cambio de divisa en tiempo real, la cual es tomada de una fuente de servicio de datos abiertos que provienen de una entidad confiable como lo es el Banco Central Europeo. Así mismo, estas dos tareas se resumen como servicios en la nube.

  Tanto el cliente como los servicios tienen los suficientes recursos para realizar las distintas tareas mencionadas.

- **El cliente tiene características de escalabilidad**

  La escalabilidad del cliente es tanto en plataforma, servicios y usuarios. Al referirse a plataforma, la aplicación al haber sido desarrollada con Apache Cordova permite ser desplegada rápidamente a otros entornos como iOS o navegadores web con el mismo código fuente, este último caso fue presentado en la implementación del proyecto para observar la interacción de los microservicios utilizados por el asistente en un navegador web. El segundo caso, la escalabilidad de servicios se da ya que la aplicación fue desarrollada con un enfoque de microservicios, permitiendo así la integración otros que funcionen a la par sin afectar el funcionamiento actual. Y la escalabilidad de usuarios, se da porque la aplicación puede ser utilizada por varias personas, aparte de los usuarios objetivos, ya que no tiene costo y puede ser instalada con facilidad en móviles con sistema operativo Android.

- **Los servicios en la nube soportan altos picos de trabajo**

  Primero se debe mencionar que el microservicio de identificación de billetes al no ser requerido constantemente, trabaja bajo peticiones, hasta entonces tiene un estado de espera (standby) y cuando se ejecuta la aplicación móvil es levantado.

  Una vez levantado, el servicio tiene la capacidad de responder a múltiples peticiones a la vez ya que se utiliza el concepto serverless computing en la implementación del microservicio con AWS Lambda el cual permite centrarse en la funcionalidad del código a ejecutarse y no del aprovisionamiento de los recursos necesarios para el despliegue ya que de eso se encarga el proveedor de servicios en la nube, en este caso AWS. Permitiendo así, un gran número de solicitudes desde distintos teléfonos móviles.

Otro objetivo logrado fue el estudio de la situación actual de aplicaciones enfocadas al reconocimiento de billetes presentada en el estado del arte. Algunos trabajos similares permiten solo reconocer un único tipo de divisa, mientras que algunas aplicaciones similares están enfocadas a una sola plataforma o tienen un costo promedio de 10 €. Con el asistente desarrollado se tiene ventajas frente a estas aplicaciones como las siguientes:

- El asistente permite reconocer dos tipos de divisas, dólares americanos y euros. Y el clasificador puede ser reentrenado para adaptarse a más divisas.

- La aplicación es desplegada en Android, pero permite desplegarse rápidamente en sistemas iOS y navegadores web sin necesidad de reescribir el código.
- En cuanto a costo es accesible para quien la requiera ya que es gratuito porque se han utilizado herramientas de la nube gratuitas y con posibilidad de escalar.
- El asistente tiene la posibilidad de integrar otros servicios adicionales sin afectar el funcionamiento actual.

5.2. Trabajos Futuros

El asistente accesible fue planteado como una prueba de concepto y como se pudo apreciar se ha trabajado por módulos, permitiendo así incluir más características o mejorar las existentes en este sistema. Por ejemplo, la red neuronal puede ser reentrenada con un nuevo conjunto de billetes de distinta(s) divisa(s) ya que el clasificador desarrollado permite tener más opciones de clasificación al realizar un nuevo entrenamiento. Y con el servicio actual de cambio de divisa se puede obtener cualquier otra que se integre.

En la aplicación se pueden integrar más sensores del teléfono móvil para obtener información como por ejemplo la localización del usuario y poder determinar los puntos de mayor uso de la aplicación como podría ser un supermercado o tienda.

El uso de un entorno web da la posibilidad de incluir temas de seguridad web como por ejemplo el uso de API Keys, JSON Tokens, etc., con el fin de estudiar las vulnerabilidades en el intercambio de información entre la aplicación y los servicios en la nube.

Chapter 6

Introduction

"Let them rise to the challenge of Sustainable Development Goals and act, not out of self interest, but out of common interest. I am very aware of the preciousness of time. Seize the moment, act now."
— Stephen Hawking

Today we live in a world in which the development and presence of technology in our lives increase minute by minute. The goal of wanting to communicate between different parts of the planet through technological devices in real time has resulted in a fully connected world in which almost anywhere you have access to the largest data network, the Internet.

Specifically, the Internet of Things (IoT) is evolving in such a way that, by 2024, it is estimated that there will have 4 billion devices connected to the Internet, only in IoT environments. This figure already indicates a huge number of connected devices, if we add other as many as mobile phones, the figure rises to 8.9 billion (Cerwall, 2015).

IoT is understood as those solutions based on processing locally acquired information through sensors, such as cameras, in an intelligent system. These systems frequently centralize information, managing data from various devices, allowing you to decide to combine them with each other or with data from other sources.

This has allowed the development of countless applications, some of them focused on social assistance; to mention some are TapTapSee which is used for object recognition, Boop Light Detector allows to detect if the light of a room is still on, NantMobile Money Reader is used to identify the value of any legal tender from different countries, among others. In this way, people with disabilities such as visual disabilities have become more integrated into society thanks to the advancement of technology and the use of applications such as those mentioned.

6.1. Motivation

According to the World Health Organization (WHO), it is estimated that approximately 1.3 billion people worldwide live with some form of near or far vision impairment. With respect to distance vision, 188.5 million people have moderate visual impairment, 217 million have moderate to severe visual impairment and 36 million are blind. On the other hand, 826 million people suffer from a deficiency of near vision (OMS, 2018). In Spain, according to information from the National Organization for the Blind of Spain (ONCE),

72,231 people are affiliated and registered with visual impairment, of which 14.24% are blind and the rest have a milder visual impairment as seen in Figure 6.1 (ONCE, 2019).

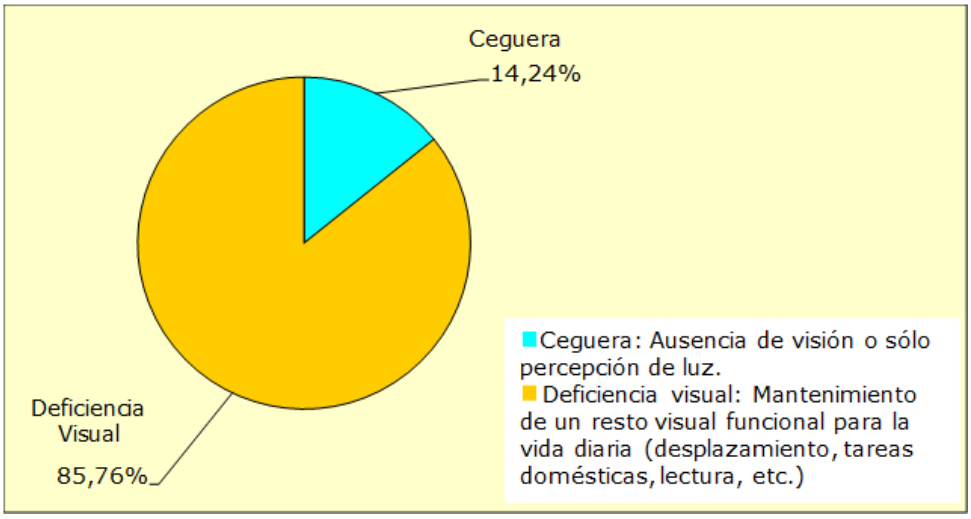

Figure 6.1: Distribution of total ONCE affiliates by type of loss. Source: ONCE (2019).

Visually impaired people may encounter problems on a daily basis, for example when it comes to distinguishing banknotes, especially in the case of foreign currencies when making purchases or simply having money on hand.

According to an article published by the ONCE (2016), there are some applications with good acceptance by visually impaired people, since in some way or another they help in their day to day; NantMobile Money Reader highlights among them that it is an application for money recognition for iOS mobile devices. This informative application of the value of any legal tender of different countries and is available in the App Store for a cost of 13.99 €.

It should be mentioned that some mobile applications are not designed for blind people as end users because they do not have accessibility features for their use. That is to say, the handling or navigation is tedious and applications that provide these characteristics usually have an average cost of 10 €.

Taking into account the aforementioned, the idea of developing an assistant that will identify banknotes, specifically euros and US dollars has been raised, along with its value for mobile devices with the Android operating system which is focused on blind people. So the application will cover minimum accessibility requirements for end users. In addition, once the value of the ticket has been identified, the application will allow knowing the current currency exchange value and as a notable difference from other applications it will be free, in addition to being adaptable to other platforms or operating systems.

6.2. Objetives

The main objective of this project is to develop a mobile accessible assistant with Android operating system, which allows blind people to identify bills (dollars or euros) with their currency exchange in real time.

In addition, other relevant objectives for the implementation of this project are presented, which are highlighted below:

- Study and document the current situation of applications focused on banknote recog-

nition.

- Select and describe the device to be used, which includes the features for working in an IoT environment.
- Study the technologies that can be used to develop IoT applications for mobile devices and that can scale.
- Investigate, develop and implement an artificial intelligence model for the recognition of banknotes, euros and US dollars, through images.
- Investigate and add a service for currency exchange according to market information in real time.
- Define which devices or services are going to compute the information.
- Select which infrastructure or service is going to implement the classification model.
- Integrate the technologies used so that they work together.
- Determine the end users for the application.
- Develop and implement the application to work on the IoT device with Android operating system.

6.3. Workflow

To carry out the project, the following workflow has been followed:

- **Analysis of the problem.** In this first phase it is about understanding the problem, by compiling different sources of information and current state of the art, linked to the realization of the project as well as the requirements of the potential users of it to fix the functionality of the project to implement.
- **Design of the software architecture.** For this phase, the modules into which the application is to be divided, as well as the technologies necessary for the development of each one, will be decided.
- **Study of the technologies to be used.** With the designed architecture, to will have to study and configure the necessary technologies to implement this project.
- **Software development.** In this stage, all the modules resulting from the study of the third phase will be implemented and linked.
- **Implementation.** In this phase, the application will be deployed to work on the selected IoT device and allow interaction with end users, determining what improvements can be implemented for future work.
- **Drafting of the memory and documentation of the project.** This phase is carried out in parallel to all the previous ones, in this way all the information obtained during the development is compiled for the drafting of the documentation.

6.4. Structure of this document

This section presents a general view of this document, which is organized into different chapters, with the following structure.

Chapter 1 is the introduction to the project and describes the motivation, objectives and structure of the project.

Chapter 2 illustrates the state of the art on the project, which in the first part describes concepts on which this project is based and requires knowledge. It is also presented to end users and how important is an application with an accessible approach, zero cost, deployable on other platforms and with its own intelligence. Finally, projects similar to banknote recognition with their characteristics are described.

Chapter 3 presents the different resources, technologies and platforms used based on the concepts seen in Chapter 2. All this allows the IoT infrastructure of the project to be formed, in this way in the final part it is described how the cloud services are associated with the client that will be the application used by the target users.

Chapter 4 describes the implementation of the project, showing how the entire project was developed from initial requirements such as technologies, platforms, etc., to the resources implemented to deploy the banknote identification microservice. the classification model and how currency exchange information is obtained. In addition, the application for the mobile device with the Android operating system is presented as it will be used by blind people who are the target user of this project.

The Chapter 5 presents a summary of the work carried out and what has been achieved, commenting on the conclusions obtained from the development, the problems encountered and proposals for possible future work.

Chapters 6 and 7 are presented in English. These contain the same information from Chapter 1 and 5, respectively.

Chapter 7

Conclusions and Future Work

"Just because something doesn't do what you planned it to do doesn't mean it's useless."
— Thomas Edison

This chapter focuses on presenting the conclusions and future work in relation to this project, making a retrospective of what has been done and taking into account that everything is under the IoT paradigm. Regarding the objectives set at the beginning, all have been achieved.

7.1. Conclusions

The main objective of this project was to develop an accessible assistant that allows banknote identification and currency exchange in real time with IoT concepts and technologies. This assistant was implemented for mobile phones with Android operating system using TalkBack as an accessibility tool to facilitate use and interaction with the user. Cloud resources and services were also integrated within this assistant.

It should be mentioned that the accessible assistant developed is a proof of concepts, since it allowed the integration of various IoT technologies and tools such as sensors, artificial intelligence, microservices, multiplatform applications, cloud services, etc.

Another objective of the project was to implement an artificial intelligence model that allows banknote identification. For this reason, the model was developed based on resnet50, a pre-trained neural network with 14 million images and provided in the fast.ai library. To adjust the network to the problem, a training set was formed with 440 banknote images with different values between euros and dollars, thus achieving high accuracy and precision, around 95%. In addition, the model was tested with a test set other than the initial fit, obtaining a hit in most of its predictions as shown in Tables 4.1 and 4.3. There were cases in which the banknote could not be correctly determined, for this reason, prediction alternatives such as those shown in Tables 4.2 and 4.4 were shown.

Also, throughout this document I presented the design and implementation of the infrastructure for the development of the accessible assistant, which uses IoT concepts presented in the state of the art and a very common IoT device in the daily life of people which is the mobile phone to give it an added value by exploiting the resources that comes with it. Based on the implemented infrastructure I conclude the following:

- **The execution of the tasks have been efficiently distributed:**

The mobile phone was used as an IoT device that executes the assistant or the developed application which interacts with the camera to photograph the different banknotes to be processed. In addition, it presents the information obtained from identification and currency exchange, with the help of the TalkBack accessibility assistant that reproduces the displayed text by voice. All of this is summed up as a customer.

On the other hand, there is the banknote identification task which is processed in the cloud with the help of AWS Lambda, which executes the classification model stored in S3, makes the predictions and sends the resulting information. There is also the task of knowing the currency exchange in real time, which is taken from a source of open data service that comes from a reliable entity such as the European Central Bank. Also, these two tasks are summarized as cloud services.

Both the client and the services have sufficient resources to carry out the different tasks mentioned.

- **The client has scalability features**

  The scalability of the client is both in platform, services and users. When referring to platform, the application having been developed with Apache Cordova allows it to be quickly deployed to other environments such as iOS or web browsers with the same source code, this last case was presented in the implementation of the project to observe the interaction of the microservices used by the assistant in a web browser. The second case, the scalability of services is given because the application was developed with a microservices approach, thus allowing the integration of others that work at the same time without affecting the current operation. And the scalability of users, is given because the application can be used by several people, apart from the target users, since it has no cost and can be easily installed in mobiles with Android operating system.

- **Cloud services support high work peaks**

  First, it should be mentioned that the banknote identification microservice, as it is not constantly required, works under requests, until then it has a waiting state (standby) and when the mobile application is executed it is lifted.

  Once up, the service has the ability to respond to multiple requests at the same time since the serverless computing concept is used in the implementation of the microservice with AWS Lambda, which allows focusing on the functionality of the code to be executed and not on the provisioning of the resources required for deployment since that is done by the cloud service provider, in this case AWS. Thus, allowing a large number of requests from different mobile phones.

Another objective achieved was the study of the current applications focused on the recognition of banknotes presented in the state of the art. Some similar jobs only allow to recognize a single type of currency, while some similar applications are focused on a single platform or have an average cost of 10 €. With the developed assistant there are advantages over these applications such as the following:

- The assistant allows you to recognize two types of currencies, US dollars and euros. And the classifier can be trained to fit more currencies.
- The application is deployed on Android, but it can be quickly deployed on iOS systems and web browsers without the need to rewrite the code.

- In terms of cost, it is accessible for those who require it since it is free because free cloud tools have been used and with the possibility of scaling.
- The assistant has the possibility to integrate other additional services without affecting the current operation.

7.2. Future Work

The accessible assistant was proposed as a proof of concept and, as it could be seen, it has been worked in modules, thus allowing to include more features or improve those existing in this system. For example, the neural network can be retrained with a new set of banknotes of different currency (s) since the developed classifier allows to have more classification options when performing a new training. And with the current currency exchange service you can get any other that is integrated.

More mobile phone sensors can be integrated into the application to obtain information such as the location of the user and to be able to determine the points of greatest use of the application, such as a supermarket or store.

The use of a web environment gives the possibility of including web security issues such as the use of API Keys, JSON Tokens, etc., in order to study the vulnerabilities in the exchange of information between the application and the services in the cloud.

Bibliografía

Aguado, J. y Estrada, F. Guía de accesibilidad de aplicaciones móviles, 2017. Disponible en https://administracionelectronica.gob.es/pae_Home/pae_Estrategias/pae_Accesibilidad/pae_documentacion/pae_eInclusion_Accesibilidad_de_apps.html (último acceso, Junio, 2020).

Android. Build more accessible apps, 2020. Disponible en https://developer.android.com/guide/topics/ui/accessibility (último acceso, Junio, 2020).

Arteaga, G. y Elizalde, C. *Discapacidad Visual*. Proyecto Fin de Carrera, Universidad de las Américas Puebla, 2007.

AWS. AWS Serverless Application Repository, 2020a. Disponible en https://aws.amazon.com/es/serverless/serverlessrepo/ (último acceso, Junio, 2020).

AWS. AWS Lambda, 2020b. Disponible en https://aws.amazon.com/es/lambda/ (último acceso, Junio, 2020).

AWS. Tutorial: Deploying a Hello World Application, 2020c. Disponible en https://docs.aws.amazon.com/es_es/serverless-application-model/latest/developerguide/serverless-getting-started-hello-world.html (último acceso, Junio, 2020).

AWS. About Amazon Web Services, 2020d. Disponible en https://aws.amazon.com/about-aws/ (último acceso, Junio, 2020).

AWS. AWS Identity and Access Management (IAM), 2020e. Disponible en https://aws.amazon.com/es/iam/ (último acceso, Junio, 2020).

AWS. Amazon S3, 2020f. Disponible en https://aws.amazon.com/es/s3/ (último acceso, Junio, 2020).

AWS. Amazon API Gateway, 2020g. Disponible en https://aws.amazon.com/es/api-gateway/ (último acceso, Junio, 2020).

AWS. Installing the AWS SAM CLI on Linux, 2020h. Disponible en https://docs.aws.amazon.com/serverless-application-model/latest/developerguide/serverless-sam-cli-install-linux.html (último acceso, Junio, 2020).

Ballesteros, C. *Conectografía de la gripe aviar: herramientas para predecir la difusión de la enfermedad*. Trabajo de fin de máster, Universidad Complutense de Madrid, 2019.

Barr, J. AWS Named as a Leader in Gartner's Magic Quadrant for Cloud AI Developer Services, 2020. Disponible en https://aws.amazon.com/blogs/aws/aws-named-as-a-leader-in-gartners-magic-quadrant-for-cloud-ai-developer-services/ (último acceso, Junio, 2020).

Batra, Y. The Dance of Data: How to Choreograph your Databases with their Microservices, 2020. Disponible en https://www.nagarro.com/en/blog/choreograph-databases-with-microservices (último acceso, Mayo, 2020).

Caschetto, V. *Integración de servicios y tecnologías IoT: Detección de vehículos y predicción de niveles de tráfico*. Trabajo de fin de máster, Universidad Complutense de Madrid, 2019.

Cerwall, P. Ericsson mobility report. Informe técnico, Ericsson, 2015.

Cordova, A. Overview, 2020. Disponible en https://cordova.apache.org/docs/en/9.x/guide/overview/index.html (último acceso, Junio, 2020).

Docker. Get Started - Overview, 2020a. Disponible en https://docs.docker.com/get-started/overview/ (último acceso, Junio, 2020).

Docker. Docker Hub Quickstart, 2020b. Disponible en https://docs.docker.com/docker-hub/ (último acceso, Junio, 2020).

Estorach, V. Accesibilidad móvil: Lo que necesitas saber, 2020. Disponible en https://www.vanessaestorach.com/accesibilidad-movil/ (último acceso, Junio, 2020).

Fast.ai. fast.ai - Making neural nets uncool again, 2020. Disponible en https://www.fast.ai/ (último acceso, Abril, 2020).

Fowler, M. y Lewis, J. Microservices, 2014. Disponible en https://martinfowler.com/articles/microservices.html (último acceso, Mayo, 2020).

Google. Google lens. 2020. Disponible en https://lens.google.com/ (último acceso, Mayo, 2020).

Hayaku. Cash reader tool. 2019. Disponible en https://cashreader.app/es/ (último acceso, Mayo, 2020).

Howard, J. y Gugger, S. Fastai: A layered api for deep learning. *Information*, vol. 11(2), página 108, 2020. ISSN 2078-2489.

LookTel. Nantmobile money reader. 2020. Disponible en http://www.looktel.com/moneyreader (último acceso, Mayo, 2020).

Martín, L. *Evaluación temporal del entrenamiento e inferencia de redes neuronales sobre plataformas hardware heterogéneas*. Trabajo de fin de máster, Universidad Complutense de Madrid, 2019.

Matich, D. J. *Redes Neuronales: Conceptos Básicos y Aplicaciones*. Universidad Tecnológica Nacional – Facultad Regional Rosario, 2001.

Microsoft. Seeing ai. 2020. Disponible en https://www.microsoft.com/en-us/ai/seeing-ai (último acceso, Mayo, 2020).

Moretti, I., Jorge, J., Amado, J., Puntillo, D. y Caniglia, C. Software libre de reconocimiento de billetes para personas en situación de discapacidad visual. En *2º Simposio Argentino sobre Tecnología y Sociedad*. 2017.

Ng, A. Structuring Machine Learning Projects, 2020. Disponible en `https://www.coursera.org/learn/machine-learning-projects` (último acceso, Junio, 2020).

OMS. Ceguera. Organización Mundial de la Salud, 2018. Disponible en `https://www.who.int/es/news-room/fact-sheets/detail/blindness-and-visual-impairment` (último acceso, Mayo, 2020).

ONCE. *Discapacidad visual y autonomía personal. Enfoque práctico de la rehabilitación*. Organización Nacional de Ciegos de España, primera edición, 2011.

ONCE. Tus 11 apps indispensables. Organización Nacional de Ciegos de España, 2016. Disponible en `https://www.once.es/blog/articulo/2016-10-28/tus-11-apps-imprescindibles` (último acceso, Mayo, 2020).

ONCE. Datos de afiliados. Organización Nacional de Ciegos de España, 2019. Disponible en `https://www.once.es/dejanos-ayudarte/afiliacion/datos-de-afiliados-a-la-once` (último acceso, Mayo, 2020).

Ortiz, J., Toapanta, M., Chavez, Y. y Lino, K. Usability and accessibility: Study guides for applications on mobile devices. *Dominio de las Ciencias*, vol. 3, páginas 1181 – 1209, 2017. ISSN 2477-8818.

Overflow, S. Insights >Developer Survey: Most Popular Technologies, 2020. Disponible en `https://insights.stackoverflow.com/survey/2020/` (último acceso, Junio, 2020).

Pytorch. From Research to Production, 2020. Disponible en `https://pytorch.org/` (último acceso, Junio, 2020).

Roberts, M. Serverless Architectures, 2018. Disponible en `https://martinfowler.com/articles/serverless.html` (último acceso, Junio, 2020).

Salcedo, F. *Identificación de papel moneda mediante reconocimiento de patrones*. Proyecto Fin de Carrera, Universidad de Lleida, 2017.

Sosinsky, B. *¿Qué es la nube? El futuro de los sistemas de información.*. Anaya Multimedia, 2011.

Tamayo, K. *Sistema de reconocimiento de billetes para personas con discapacidad visual mediante visión artificial*. Proyecto Fin de Carrera, Universidad EIA - Envigado, 2018.

TensoFlow. Why TensorFlow, 2020. Disponible en `https://www.tensorflow.org/` (último acceso, Junio, 2020).

para los negocios Cámara de Valencia, T. Caminar con éxito hacia la Industria 4.0: Capítulo 14 – Dispositivos (I) Internet de las cosas (IoT), 2018. Disponible en `https://ticnegocios.camaravalencia.com/servicios/tendencias/caminar-con-exito-hacia-la-industria-4-0-capitulo-14-dispositivos-i-internet-de-las-cosa` (último acceso, Junio, 2020).

Väin, M. Foreign exchange rates API with currency conversion, 2020. Disponible en `https://exchangeratesapi.io/` (último acceso, Junio, 2020).

WAB. Blindness: Challenge and achievement. World Access for the Blind, 2012. Disponible en https://waftb.net/blindness-challenge-and-achievement (último acceso, Mayo, 2020).

Wikipedia. Computación en la nube, 2020. Disponible en https://es.wikipedia.org/wiki/Computaci%C3%B3n_en_la_nube (último acceso, Junio, 2020).

Apéndice A

Apéndice A. Requerimientos iniciales

A.1. Apache Cordova con Docker

A continuación se presenta como se ha preparado el entorno para el desarrollo de la aplicación, ya que Apache Cordova se utilizará como una plataforma portable y para ello se requiere de la tecnología de contenedores de Docker. Para esto se procede a instalar Docker y posteriormente a montar una imagen de Apache Cordova.

A.1.1. Instalación de Docker

Para instalar Docker se siguen los siguientes pasos:

```
sudo apt update
sudo apt install apt-transport-https ca-certificates curl \
software-properties-common
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | \
sudo apt-key add -
sudo add-apt-repository "deb [arch=amd64] \
https://download.docker.com/linux/ubuntu focal stable"
sudo apt update
apt-cache policy docker-ce
sudo apt install docker-ce
sudo systemctl status docker
```

Los comandos de Docker por lo general se deben ejecutar con sudo ya que solo puede ser ejecutado por el usuario root o un usuario del grupo Docker. Para evitar esto se debe agregar el usuario al grupo Docker con los siguientes comandos.

```
sudo usermod -aG docker ${USER}
su - ${USER}
id -nG
```

A.1.2. Imagen de Apache Córdova montada en Docker

Para utilizar Apache Cordova con Docker, primero se debe buscar una imagen dentro de la página de Docker Hub con las siguientes palabras clave "docker-android-cordova" y se despliegan varias opciones. La imagen seleccionada ha sido **vgaidarji/docker-android-cordova**, ya que se ha tomado en cuenta las características a fecha 2 de Mayo de 2020:

- Cantidad de descargas: 3.2k
- Última actualización: 1 día atrás
- Tipo de sistema operativo de la imagen: Ubuntu

En la máquina virtual se proceden a ejecutar los siguientes comandos para descargar la imagen y posteriormente ejecutar el contenedor.

```
docker pull vgaidarji/docker-android-cordova
```

```
docker run -t -dit -v /home/francis/Desktop/tfm/frontend:/bitrise/src/tfm
vgaidarji/docker-android-cordova bash
```

El segundo comando ejecutado, permite compartir una carpeta del directorio local con el directorio dentro del contenedor.

Se debe ejecutarlo para guardar el desarrollado realizado de manera local (fuera del contenedor) ya que los contendedores se enfocan en proporcionar los recursos necesarios para el funcionamiento de la plataforma más no de almacenar información respecto al desarrollo que se realice con cada uno.

El comando indicado solo se debe ejecutar una sola vez, a continuación se muestra como se puede activar el contenedor antes ejecutado para utilizarlo cuando se requiera.

- Con el comando `docker ps` se puede conocer las imágenes ejecutadas.
- Con el comando `docker exec -it idContenedor bash` se ejecuta de modo iterativo al contenedor que se requiere.

A.2. Recursos de AWS en local

A.2.1. Instalación de SAM

La instalación se ha hecho en base a la guía que se encuentra en AWS (2020h). Esta guía indica que AWS SAM requiere **AWS CLI** y **Homebrew**.

A continuación se presentan los comandos para instalar Homebrew.

```
sh -c "$(curl -fsSL \https://raw.githubusercontent.com/Linuxbrew/docker/
master/install.sh)"
```

```
test -d ~/.linuxbrew && eval $(~/.linuxbrew/bin/brew shellenv)
```

```
test -d /home/linuxbrew/.linuxbrew && eval $(/home/linuxbrew/.linuxbrew/
bin/brew shellenv)
```

```
test -r ~/.bash_profile && echo .^eval \$($(brew -prefix)/bin/brew \
shellenv)">>~/.bash_profile
```

```
echo "eval \$($(brew -prefix)/bin/brew shellenv)">>~/.profile
```

Una vez listo el entorno Homebrew, se procede a instalar AWS SAM.

```
brew tap aws/tap
```

```
brew install aws-sam-cli
```

En base a la experiencia de este proyecto es recomendable utilizar los *templates* que brinda AWS SAM, de esta manera se tendrá un inicio para el desarrollo de cualquier aplicación y conforme se avanza se pueden quitar o agregar más funciones o recursos según se requiera. Para este proyecto se ha utilizado un template que viene asociado con PyTorch, este ejemplo se puede obtener al ejecutar el siguiente comando.

```
wget https://github.com/fastai/course-v3/raw/master/docs/production/aws-lambda.zip
unzip aws-lambda.zip
```

A.2.2. Librerías de Python

Un lenguaje de programación muy importante dentro de este proyecto es Python ya que permite el desarrollo del microservicio para reconocimiento de billetes. El sistema operativo indicado al inicio ya cuenta con Python, específicamente con la versión 3.7.2, pero se requieren algunas librerías adicionales para trabajar con AWS SAM y son agregadas con ayuda de **pip**, el cual se instala con el siguiente comando.

```
sudo apt install python3-pip
```

Ahora se procede a instalar todas las librerías adicionales que son requeridas tanto por AWS SAM para su instalación como para el desarrollo del microservicio. Se puede instalar todas estas librerías al ejecutar un comando, pero para ello se debe crear un archivo con extesión `.txt` el cual se denominará *requirements* y por dentro contendrá lo siguiente:

Código A.1: Contenido del archivo requirements.txt.

```
1 fastai
2 python-multipart
3 starlette
4 aiofiles
5 aiohttp
```

Una vez instalado pip y creado el archivo requirements.txt, se ejecuta el comando

```
pip3 install -r requirements.txt
```

A.2.3. Configuración de AWS CLI en entorno local

AWS CLI es la interfaz con línea de comandos de AWS. Se requiere tener instalada para que AWS SAM pueda desplegar el desarrollo del microservicio hacia la nube.

```
sudo apt install awscli
```

Una vez instalada se conecta con el entorno local con ayuda de las credenciales **AWS Access Key ID** y **AWS Secret Access Key** creadas en la sección 4.2.1, además se define la región en la que se quiera trabajar. Esto se observa en la la Figura A.1.

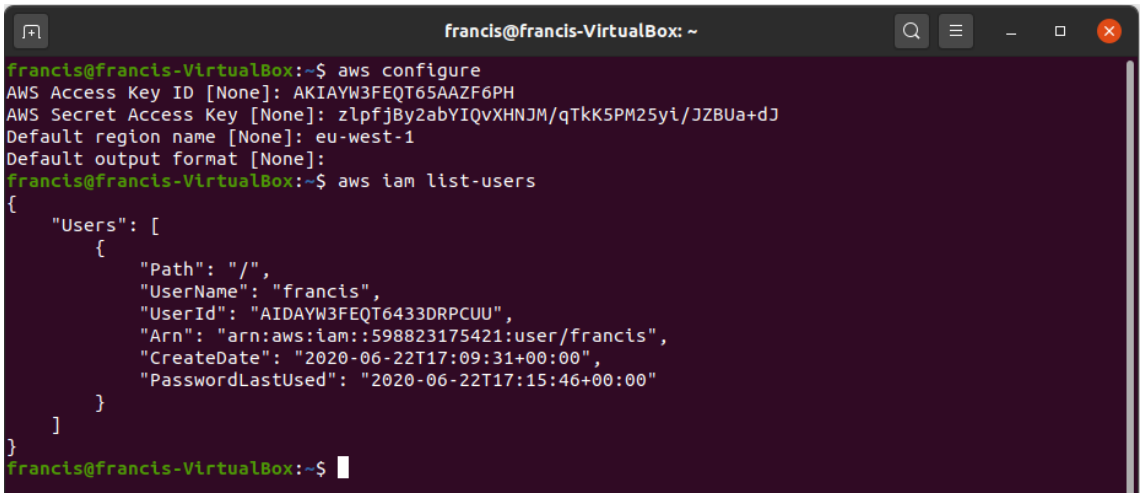

Figura A.1: Conexión local con AWS.

Apéndice B

Apéndice B. Códigos

B.1. Implementación del modelo de clasificación

La creación y ejecución del modelo se puede observar en el siguiente enlace `https://github.com/francisandres7/tfmUcm/blob/master/Modelado/tfm.ipynb`.

B.2. Implementación de la función Lambda

Los códigos para la función Lambda desarrollada con AWS SAM se encuentran en el siguiente enlace `https://github.com/francisandres7/tfmUcm/blob/master/FunciónLambda/`.

B.3. Implementación del cliente

Los códigos para la aplicación realizada en Cordova se encuentran en el siguiente enlace `https://github.com/francisandres7/tfmUcm/blob/master/Aplicación`.